UNIVERSITY OF CALIFORNIA
SANTA CRUZ

ROBUST LOOP-FREE ON-DEMAND ROUTING IN AD HOC NETWORKS

A dissertation submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Hari Rangarajan

June 2006

The Dissertation of Hari Rangarajan is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Katia Obraczka

Professor Anujan Varma

Lisa C. Sloan
Vice Chancellor for Research and
Dean of Graduate Studies

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2006		**00-06-2006 to 00-06-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Robust Loop-Free On-Demand Routing in Ad Hoc Networks	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**223**	

Contents

List of Figures

List of Tables

Abstract

Robust Loop-free On-demand Routing

in Ad hoc Networks

by

Hari Rangarajan

In this thesis, we explore new techniques for robust, efficient, loop-free on-demand routing in Mobile Ad hoc Networks (MANETs) using the same information (i.e., topology information, sequence numbers, etc.,) used in prior on-demand routing protocol proposals.

We provide new insights into the robustness of protocols based on destination-sequence numbers when operating with node failures and loss of routing information, and present a new destination-sequence number framework that works correctly even with failure conditions. We also show how destination-sequence numbers can be manipulated as routing labels for improving performance, rather than being strictly treated as time-stamps.

We present two different routing approaches that exploit the topology information that can be collected on-demand during the route request flood search. The first approach translates topology information into labels that are then stored at nodes in strict ”lexicographic” ordering along any successor path to a destination. Loop-freedom is maintained by allowing nodes to only pick ”smaller” labels. The second approach maintains a list of topology information that should not be trusted in-addition to the known path to a destination. Using the notion of trusted topology, nodes can always make routing decisions only with the correct topology information, which ensures loop-freedom.

We conclude our research with a new on-demand routing technique that exploits the route request flood search process, which is an integral part of any on-demand routing protocol. Without requiring any additional mechanisms or information, we use the uniqueness of route requests to realize a on-demand routing framework. We, then, present extensions to the basic framework to improve performance using more information that can be collected during the flood search process.

Through extensive simulations, we show that all our new proposed approaches perform better than the current state-of-the-art MANET protocols prescribed by the IETF working group.

To my mother

Acknowledgments

I believe that a lot of my life's experiences both personal and professional had a definite say in the shaping of this thesis. So, first, I would like to thank everyone who influenced or contributed to my life in that regard – perhaps, they were destined to do so by the cosmic balance of nature (or driven by an ad hoc sequence of events?).

Of these people, no least, I would have to start off by expressing my heart-felt gratitude towards my Adviser, Prof. J.J. Garcia-Luna-Aceves, who has always been an excellent role-model to strive to emulate. I consider it as a great privilege to have been associated with him and a great pleasure to learn from him. I would have to specially thank him for persisting with me at times even as I went through my "phases" in life. I will always be left wondering in life if I could have gone on to finish without him as my adviser.

Next in line, I would have to thank my mother, who would rank as the single most influential person in my life. She has been my best well-wisher and I'm very grateful for everything that she has done for me. I would also like to thank my brother who has been around here during the last year.

In the past years, I have had the opportunity to be associated with many people here at UCSC. A limited list would feature Saro, Soumya, Ramesh, Venkatesh, Yu, Marcelo, Marco, Marc, Wenyi, Zhenjiang, Ravindra, and Rahul; I'm sure I have learnt a thing or two about research or life in general from them. I would also like to mention a special thanks to Vishwanath Venkatraman for many conversations reflecting on life (and the PhD too, I think?). I'm definitely missing out many other people who did contribute significantly but I would still like to

thank them all if they believe that they should be on this list.

The text of this dissertation includes material that has been previously published [4, 26, 34, 14, 5, 15]. Prof. J.J. Garcia-Luna-Aceves, listed as the co-author in these publications directed and supervised the thesis research that forms the basis for this dissertation.

This work was funded in part by the Baskin Chair of Computer Engineering at UCSC, the National Science Foundation under Grant CNS-0435522, the UCOP CLC under grant SC-05-33, and by the U.S. Army Research Office under grant No. W911NF-05-1-0246.

Chapter 1

Introduction

A mobile ad hoc network (MANET) is characterized by a set of mobile nodes capable of communicating over wireless media and establishing a network without the need for a pre-existing infrastructure. The routing protocol of a MANET enables nodes to deliver data across multiple hops, and has received considerable attention from the research community for more than two decades ([19, 13, 8, 25, 28, 29, 2]). Routing protocol design for a MANET presents a unique challenge incontrast to wired networks. The fundamental design challenge for a routing protocol in a MANET is the limited bandwidth offered by the wireless media which necessitates the use of minimal control signalling to attain loop-free routing. In addition, the dynamic characteristics of MANETs, coupled with the physical characteristics of the wireless media which cannot guarantee reliable delivery, demand that routing protocols be robust (i.e., produce correct routing tables in a finite time) in the presence of node and link failures, network partitions and re-constitutions, and nodes rejoining a network. Routing in MANETs can be classified with two major approaches: Pro-active approaches maintain routing information for

all destinations, regardless of whether traffic exists for them. Reactive (on-demand) approaches maintain routing information for only those destinations for which traffic exists, and rely on flood search mechanisms to establish routes to "active" destinations. On-demand protocols are very attractive in scenarios with high mobility, and traffic patterns in which source nodes pick a few other nodes as destinations. All on-demand protocols attempt to maintain loop-free routing tables, because routing loops could lead to excessive signaling overhead trying to repair loopy routes. The three basic approaches used in on-demand protocols to eliminate loops are nodal synchronization, sequence numbers, and source routing using topology information. Examples of nodal synchronization include the temporally-ordered routing algorithm (TORA) [37], and the Routing On-demand with Acyclic Multiple paths (ROAM) [36]. TORA uses a link-reversal algorithm [6] to maintain one or more loop-free paths created with a query-reply process similar to that used in all on-demand routing protocols. To terminate the algorithm, TORA requires the use of synchronized clocks. ROAM creates multiple paths similar to TORA and uses a safe condition based on distances to pick loop-free successors. When no safe successors can be found, ROAM requires synchronization spanning multiple hops. In general, protocols using nodal synchronization over one or more hops incur excessive overhead, and also require guaranteed message delivery by the MAC layer, which may not be available in MANETs.

In this thesis, we propose different schemes for loop-free routing applied in the context of on-demand routing protocols using sequence numbers and topology information. We do not investigate nodal synchronization as we assume that MAC layers used in MANETs (i.e.IEEE802.11 [35] may not be able to guarantee reliable message delivery. Our aim is to minimize control signalling, while at the same time ensuring loop-freedom and robustness of

the routing protocol. The highlight of all proposed approaches, which we introduce in the next sections, is that they only use the same information or less used in prior on-demand routing protocols. Sections 1.1 discusses related work in the area of on-demand routing protocols based on destination sequence numbers. We illustrate the short-comings of current protocols, motivate and list our contributions. Section 1.2 discusses on-demand routing protocols that use topology information. We motivate and list our contributions that provide loop-free routing using topology information without requiring source-routes. Section 1.3 motivates the need for an on-demand routing protocol that can exploit the basic route search mechanism, and lists our contributions. Chapter 6 presents our concluding remarks and discusses possible future work.

1.1 Destination Sequence numbers

Loops can be avoided in a routing protocol if nodes can determine the freshness of updates received for a destination. Sequence numbers serving as time-stamps is a very well-known approach in distributed systems to determine freshness. The Destination Sequenced Distance Vector (DSDV) routing protocol [25] was the first MANET routing protocol that proposed the use of a sequence number associated with each destination by which the updates could be time-sequenced correctly. In DSDV, all nodes pro-actively send updates periodically to install correct routes for themselves in the network.

The Adhoc On-demand Distance Vector (AODV) routing protocol [19] extended the use of sequence numbers to establish loop-free routes to a destination on-demand. In AODV, the sequence number carried in a route request (RREQ) elicits only fresher route replies (RREP)

with an equal or higher sequence number. Route errors (RERR) are sent unreliably, based on the notion that increasing the destination sequence number invalidates the route entry of all upstream nodes.

The Labeled Distance Routing (LDR) protocol [8] was recently proposed to improve on the way in which AODV uses sequence numbers, so that destinations need to respond to fewer RREQs. LDR is an on-demand routing protocol based on a dual invariant consisting of destination sequence numbers and feasible distances [7]. A feasible distance in LDR roughly corresponds to the smallest distance to a destination attained by a node for its current sequence number for the destination. The destination sequence number is used to "reset" the distance to a destination, i.e., to allow a node to accept a next hop that reported a distance larger than the node's feasible distance. Using feasible distances in LDR makes it more likely for nodes other than the destination to resolve RREQs, which improves the performance significantly [8].

In Chapter 2, we focus on the use of per-destination sequence numbers to attain loop-free routing in an on-demand routing protocol. So far, the research community has devoted considerable attention to develop loop-free protocols for MANETs using destination-based sequence numbers. However, the robustness and efficiency of such protocols has not been well-understood in MANETs, in which nodes have to delete old routing table information within finite times to scale with increasing number of nodes, and also maintain correct routing protocol operation in the presence of failures and other abrupt changes despite using unreliable MAC protocols. We provide new insight on the requirements for robust routing using destination sequence numbers when nodes may delete routing state, and present a technique for nodes to safely re-learn destination sequence numbers over an unreliable communication medium, after

reboots or deletion of routing table entries to save memory. Under such operating conditions, we show that the way in which the Ad-hoc On-demand Distance Vector (AODV) protocol handles destination-based sequence numbers can lead to looping of data packets, de-facto network partitions, and counting to infinity.

We introduce a new framework for on-demand loop-free routing based on destination sequence numbers that is robust and efficient in the presence of node failures and reboots, unreliable message transmission with unbounded queueing delays, and the loss of routing state at nodes. We apply our framework to AODV as an example of how robust sequence-number handling can be attained without compromising the efficiency of the protocol. We call this fixed version, AODV-RSN (AODV with Robust Sequence Numbering). Then, we attempt to extend the basic robust framework with a technique to improve performance, in a fashion similar to LDR. The key limitation with AODV, in which a node increases the stored sequence number for a destintion after a link failure, is that it prevents responses from nodes that are closer to the destination but have an older sequence number, even if they have a valid loop-free path to the destination. Consequently, the likelihood that the destination itself must resolve a RREQ is high, because the destination is the only node that can increase its own sequence number. This effect has been documented recently by Chakeres and Klein-Berndt [1], who describe a simplified version of AODV that eliminates sequence numbers in RREQs and requires the destination to answer each RREQ. This version of AODV, called AODVjr, was shown to have nearly the same performance as AODV in network scenarios of 25 and 50 nodes. The approach we present, called the Sequence-number Window Routing (SWR) protocol, is a more efficient instantiation of the framework based on *sequence number windows*, a technique that allows

per-destination sequence numbers to be ordered progressively along a successor path to the destination. This allows more intermediate nodes to reply to route requests correctly avoiding network-wide floods. AODV-RSN and SWR are compared via simulations with DSR, AODV and Optimized Link State Routing (OLSR) [2] protocol using networks of 50 and 100 mobile nodes. The results indicate that SWR is better than AODV on the average, while AODV-RSN's improved performance validates our claim on the correctness issues of AODV, and the necessity for immediate participation of the routing protocol after a node reboot.

1.2 Topology Information

The characteristics of MANET and the need for flooding to establish routes to a destination make the use of topology information in on-demand routing protocols very attractive. Because topology information is collected when nodes find paths to chosen destinations, it is possible to reduce the number of flood searches by taking advantage of stored topology information, and paths can also be repaired locally after link failures using the same information. Several routing schemes have been proposed for MANETs based on topology information. The optimized link state routing (OLSR) protocol [2] reduces the number of nodes that need to forward link-state updates, while other schemes communicate only partial topology information corresponding to shortest-path trees [10]) or reduce the frequency with which link-state updates are sent based on the distance to the sources of the updates [9]. However, none of these approaches prevent the creation of temporary routing loops when the "topology maps" stored at different nodes are inconsistent.

DSR is a well-known known example of using path information on demand to avoid looping in MANETs. DSR enforces loop-free routes to a destination by carrying the path traversed in the RREQ and the reverse path is then used to source route data packets to the destination. After a link failure, reliable error updates are sent to the source, so that a new route can be searched. The limitation with this approach is that data-packet headers must specify source routes to avoid loops, which incurs additional processing overhead for each data packet. Packet salvaging can be used in DSR to recover from link failures locally by re-routing data packets along an alternative source route; however, this approach may result in the formation of temporary loops, and requires a mechanism to detect data packets flowing over those loops. Recently proposed techniques [3, 21] attempt to improve the packet salvaging mechanism by using more topology information, and re-route data packets safely along alternate paths, but their recovery effectiveness is limited to the two-hop neighborhoods of nodes. An implicit source routing approach [12] has also been proposed to reduce the extra overhead of source routes in data packets by carrying flow Id's instead of the entire source route. However, this scheme still requires additional per-packet processing of IP option headers, and loops formed are detected by using the time-to-live (ttl) of the data packets.

OLIVE [43] was the first on-demand routing scheme based on link vectors (or equivalently, path information) that allowed data packets to be forwarded based solely on their destinations. Nodes collect topology information from the link vectors in route requests (RREQ) and route replies (RREP), and the freshness of a link is indicated by means of link sequence numbers. OLIVE leverages the partial topology information to send source-routed requests to recover failures at intermediate nodes, but can only pick paths of equal or lower cost. After

a link break, a source node must exchange route errors reliably with all neighbors that use the node as next hop to the destination. To cope with the possibility that route errors cannot be exchanged reliably, OLIVE requires a node to discard a data packet received from a given neighbor if the neighbor that sent the packet is in the node's path to the packet's destination.

AODV-PA [30] [16] collects partial topology information with RREQs and RREPs like DSR, OLIVE and FLR do. Loop-freedom is based on the use of per-destination sequence numbers to determine which distances to a given destination can be trusted. The Dynamic MANET On-demand (DyMO) routing [38] was a recent routing framework proposal based on destination sequence numbers, building on AODV, with the capability to add more features like path accumulation on-demand. The limitation of this approach is that it does not take advantage of path information to enforce loop-free routing. Accordingly, the performance of AODV-PA is very similar to that of AODV, and path accumulation provides only slight improvements in huge networks with many flows [16].

The Directed Incremental Routing (DIR) [4] protocol represents another approach to collecting path information together with distances. A limitation of DIR is that it does not use path information to "reset" distances when nodes reboot or delete invalid routing-table entries, which can lead to loops in routing tables.

Using path information in an on-demand routing protocol offers an alternative to using destination sequence numbers that must be frequently "reset" by the destinations, as well as to synchronizing the routing-table update steps of several nodes, which has been shown to incur even more overhead than reseting destination sequence numbers. However, none of the on-demand routing protocols based on path information proposed to date are able to support

instantaneous loop-free routing when data packets are forwarded on a hop-by-hop basis using only the addresses of their destinations. We propose two different approaches that use topology information for their routing decisions. The first approach ignores the actual correctness of the topology information, while the second maintains a notion of which topology information is correct.

1.2.1 Paths as labels

In Chapter 3, we present the Feasible Label Routing (FLR) protocol, which is the first on-demand routing protocol that uses path information only in its signaling for on-demand routing, while supporting loop-free incremental forwarding of data packets when the header of each data packet simply states the address of the intended destination. To present FLR, we extend the notion of "feasible distances" (used in the past in such protocols as the diffusing update algorithm (DUAL) and more recently the labeled distance routing (LDR) protocol [8]) to the concept of *feasible labels*, such that nodes are lexicographically ordered for a given destination according to their paths to the destination. We introduce and prove sufficient conditions for loop freedom based on feasible labels. FLR attains loop-freedom by lexicographically ordering such feasible labels along a successor path to the destination. Simulation experiments in Qualnet show that FLR and variants of FLR that use labels associated with destination sequence numbers achieve better performance than AODV, DSR, AODV-PA, and OLSR, in terms of the packet delivery ratio and average delivery latencies achieved, as well as the overhead incurred in the network.

1.2.2 Paths with untrusted topology information

All prior topology-based schemes use information about the paths that can be used to reach destinations, but do not use information about the links that cause nodes to change their routes, and the inconsistency of information regarding links that cause rerouting decisions at different nodes is what actually causes the formation of routing-table loops. In Chapter 4, we introduce the Link Vector Routing (LVR) protocol for loop-free on-demand routing in ad hoc networks. LVR differs from all prior routing protocols based on topology information in that nodes communicate and store information about those links that should not be trusted when rerouting decisions are made, in addition to the information about known links in the network. LVR achieves loop-free hop-by-hop routing without requiring reliable communication among neighbors, per-packet filtering, or source-routes or flow identifiers in data packets, as in prior on-demand link-vector proposals. LVR operates using a very simple principle idea: As in prior schemes (e.g., DSR, AODV-PA, FLR, OLIVE), routers use route requests to discover valid path information and route replies to provide the requested path information to sources of data packets. However, LVR also uses route requests and route replies to disseminate information about those links whose failure caused nodes to require rerouting and which should not be trusted when repairing routes to destinations. We prove a new sufficient condition for loop-free routing based on this dissemination of path information, and show through extensive simulations in Qualnet that the performance of LVR is far better than the performance of AODV, AODV-PA, DSR, FLR and OLSR, in terms of the packet delivery ratio and average delivery latencies achieved, as well as the overhead incurred in the network.

1.3 Source-Sequence numbers

A feature common to every on-demand routing protocols, including our protocols based on destination-sequence number and topology information, is that they share the same mechanism for searching and establishing routes. This proceeds in two distinct phases: During the *route search* phase, the network is flooded with route requests (RREQ). Each RREQ is uniquely labeled by its source by means of a *source-sequenced label* (SSL), which consists of the identifier of the source and a source sequence number that is locally unique. SSLs prevent any node from processing the same RREQ multiple times, and nodes effectively build a directed acyclic graph (DAG) rooted at the source for a source-destination pair uniquely defined by the SSL and the destination identifier. During the *route establishment* phase, RREQs received by the destination or an intermediate node with routing state for the destination are answered with route reply (RREP) messages that traverse the loop-free reverse paths along the DAG built in the route-search phase. Each node receiving a RREP establishes or updates its routing state for the destination specified in the RREP. This information is used for data forwarding and route maintenance.

Interestingly, the design of on-demand routing protocols to date has been such that the mechanisms used in the route-search phase to propagate RREQs are fairly independent of the mechanisms used during the route-establishment phase to establish and update routing-table entries for destinations. Clearly, SSLs are a necessity in any on-demand routing protocol for the identification of RREQs and their efficient processing. Furthermore, the loop-free paths that are built during the route-establishment phase must be part of the DAGs built during the route-

search phase based on SSLs. This begs the question of whether on-demand loop-free routing can be attained using SSLs during both the route search and the route establishment phases of the routing process.

In Chapter 5, we introduce the first routing framework based on source-sequenced labels (SSLs). We show how to use the minimum required information in any on-demand routing protocol (consisting of a unique (source, flooding identifier) pair carried in RREQs) to realize SSLs that can be used to identify loop-free successors for destinations, without the need for any other data (e.g., distance or path information). The labeling scheme is extended to form *Source-Sequenced Distance Labels* (SSDL) by adding the distance (hop count) to the destination to the SSL. The same scheme can be extended to other properties relating to a destination, such as paths instead of distances. Using SSDLs, we present the first on-demand loop-free routing protocol based on SSLs and distances. Using extensive simulation results, we show that simple protocol instantiations of our new framework operating in scenarios with 50 and 100-nodes under different traffic patterns outperform AODV, DSR, and OLSR.

Chapter 2

Routing using destination sequence numbers

2.1 Introduction

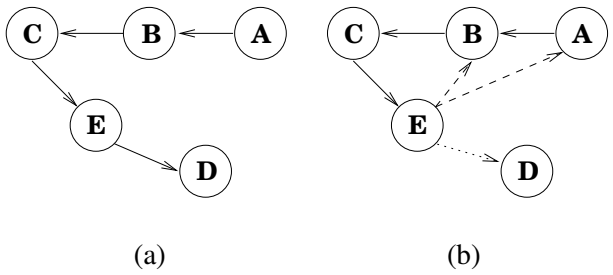
The class of on-demand routing protocols based on destination sequence numbers requires each node to store a strictly increasing sequence number for itself, and, in-addition, a sequence number for each known destination node in the network that is learnt from updates sent for the particular destination. Loop-freedom is ensured if nodes only process updates for a destination that have a higher destination sequence number than the one stored. The Adhoc On-demand Distance Vector (AODV) routing protocol [19] is the most popular on-demand routing protocol based on this approach. Although several improvements haven been proposed based on the use of destination sequence numbers, none of them delve deeper into the issues related to the maintenance of stored sequence numbers when stored information is lost due to node reboots or route table purges.

Bhargavan et al [24] identified a failure condition in AODV caused by the rebooting

of a node, and proposed the the use of the $DELETE\_PERIOD$ as a safety condition for loop-freedom in AODV in the presence of node reboots. The $DELETE\_PERIOD$ is the maximum time that a node can retain its successor for a route entry in the presence of unreliable communication, and is a property of the prevailing network conditions. The AODV specification uses this timer for managing the state of the routing table related to destination sequence numbers. Assuming that a node that reboots waits "long enough" before re-engaging in normal update activity, as prescribed by Bhargavan et al [24], the use of destination-based sequence numbers in AODV works correctly, as long as nodes maintain the last sequence number they learned for a given destination. However, to make the scheme practical in large MANETs, nodes must be allowed to delete old invalid routes after a finite time. Section 2.2 shows that AODV's handling of sequence numbers based on a $DELETE\_PERIOD$ can lead to looping, de-facto partitions and count-to-infinity behavior. Section 2.3 presents a set of necessary and sufficient conditions for loop-freedom using destination sequence numbers. Previously defined conditions for destination sequence numbers [25] are based on the assumption that either nodes never "forget" destination sequence numbers that they learn, even after reboots, or nodes are able to wait "long enough" before re-engaging in the route maintenance for a destination after losing routing state for that destination. We present a new technique that allows nodes to re-learn (i.e., reset) destination sequence numbers after reboots or after deleting route entries. The basic idea of the safe reset is to have the request from a node having no sequence number state to be answered by the destination. However, to ensure that only the latest replies received from the destination are processed, the node requesting a reset identifies a request with a strictly increasing identifier that is carried in the reply, which helps sequence the latest replies over stale replies in the network.

Section 2.4 specifies a framework that provides a robust approach to on-demand destination-based sequence numbering based on the sufficient conditions for loop-freedom introduced in Section 2.3. We show that the framework is robust and correct (i.e., it is loop-free, terminates, and converges) in the presence of node failures and reboots, unreliable message delivery, and unbounded queuing delays. The framework eliminates the necessity for arbitrary timer periods to maintain correctness and safety, while allowing nodes to both participate in routing actions immediately after reboot and purge routing-table entries at any time. We illustrate an example in which nodes re-learn sequence numbers for a destination after loss of state.

Section 2.5 presents an instantiation of AODV, which we call AODV-RSN (for robust sequence numbering) and which eliminates the problems with AODV outlined in Section 2.2. In Section 2.6, we present a more efficient instantiation of the framework, the *Sequence Number Window Routing (SWR)* protocol, which is based on the concept of creating *sequence number windows*. First, we introduce a new sufficient condition for loop-free routing that allows a sequence number to be treated as a label when assigning nodes a sequence number for a destination, without affecting the loop-freedom. The key idea in SWR is for a node to assign itself a sequence number based on the destination sequence number advertised by the successor it is using to reach a destination and by those neighbors that currently use it as the next hop for the destination. We present the motivation behind ordering sequence numbers progressively towards a destination using sequence-number windows and illustrate how more intermediate nodes are allowed to reply to route requests, instead of the destination (which is the pre-dominant case in AODV). The modifications required in terms of control messages, conditions, and message han-



(a) (b)

Figure 2.1: Issues with DELETE_PERIOD in AODV

dling are presented incrementally with respect to the basic framework. We analyze and prove the correctness of SWR, and present an example of its operation. Section 2.7 shows through simulations that the proposed new approach to the handling of sequence numbers, AODV-RSN, does not impact AODV's performance negatively, and allows nodes to participate in normal routing activities much faster than AODV after reboots. Simulation results also show that SWR performs on par or better than the other state-of-the-art MANET routing protocols (AODV, the Dynamic Source Routing (DSR) protocol [13], and the Optimized Link State Routing (OLSR) protocol [2]). The results also show the effectiveness of SWR's local repair, which uses a one-hop neighbor query in-contrast to AODV's local repair mechanism, where intermediate nodes can only perform a local-repair if they know the hop-count to the source, and must also be at a minimum distance of four hops from the source. Section 2.8 provides our concluding remarks.

2.2 Sequence Numbering Problems in AODV

We address AODV's problems with sequence numbering by way of examples assuming the use of an unreliable MAC protocol similar to IEEE802.11 DCF [35].

In AODV, a node can lose its routing state (destination sequence number) for a destination in two ways: when a node reboots or after a node deletes a route entry to save memory. We now discuss the problems associated with losing such a state in AODV.

2.2.1 Looping

Figure 2.1(a) shows an example directed acyclic successor graph (DASG) for a five-node network running AODV with nodes *A, B, C,* and E having flows to destination D. The nodes in the example have an active route entry with a valid destination sequence number for D, and Figure 2.1(b) shows the network at a subsequent time with the physical connectivity affected due to mobility. At this time, link $E - D$ goes down, and links $E - A$ and $E - B$ come up. Node E sends a RERR to advertise the unreachability of destination D, which propagates to the upstream nodes along the directed acyclic graph for D.

Consider the case of "unbounded" queueing delays or message loss. In the simplest case, if the RERR is never delivered to C, then node E can send a RREQ for D after deleting its destination sequence number, which happens after waiting for the $DELETE\_PERIOD$ [19] to expire. Nodes A or B can answer the RREQ resulting in a loop.

The AODV specification [19] defines a value for the $DELETE\_PERIOD$ equal to K times the value of the $ACTIVE\_ROUTE\_TIMEOUT$ variable, with the recommended value for K being five. However, the actual value of K is a property of the prevailing network conditions and cannot be deduced accurately. If the parameter 'K' is set too low, then it can result in loops. On the other hand, if 'K' is set too high, it can slow down the convergence of AODV or aggravate the looping problem during reboots, which we discuss next.

2.2.2 De-Facto Partitions

A node rebooting after a failure loses state about all destination sequence numbers. Assume that node E in our example of Figure 2.1(a) reboots and has to wait for a minimum of $DELETE\_PERIOD$ before it can participate in routing actions. Node E sends RERRs on receiving data packets from C, which propagate to A. New RREQs generated by A for D cannot reach D during E's reboot wait time, because E has to drop all RREQs during that time. Hence, the network appears to be partitioned, despite the physical links available in the network. This limitation can be critical in sparsely connected networks when there are no alternate paths, and can force nodes to choose sub-optimal paths for the duration of their flow, because routes are not improved pro-actively. We note that nodes that are sources or destinations of flows after a reboot are effectively partitioned from the network during the reboot wait period.

2.2.3 Counting to Infinity in AODV

The AODV specification of 2003 [19] recommends that nodes delete invalid route entries after a finite time equal to the maximum elapsed time after which a node can still send data packets to the next-hop specified in the routing table, called the $DELETE\_PERIOD$. However, as we show below, this condition is not safe.

A DASG for destination D is shown in Figure 2.2(a). We assume that all nodes run AODV correctly and have data for D at some point in time. The dotted node R between node C and node X is used to indicate a set of one or more nodes that could be along the path. All nodes are assumed to have sn_D^1 as the destination sequence number for D in their routing tables. We consider path P= $\{Y, X, \{R\}, C, B, A\}$ and we trace one of many sequences of events that can

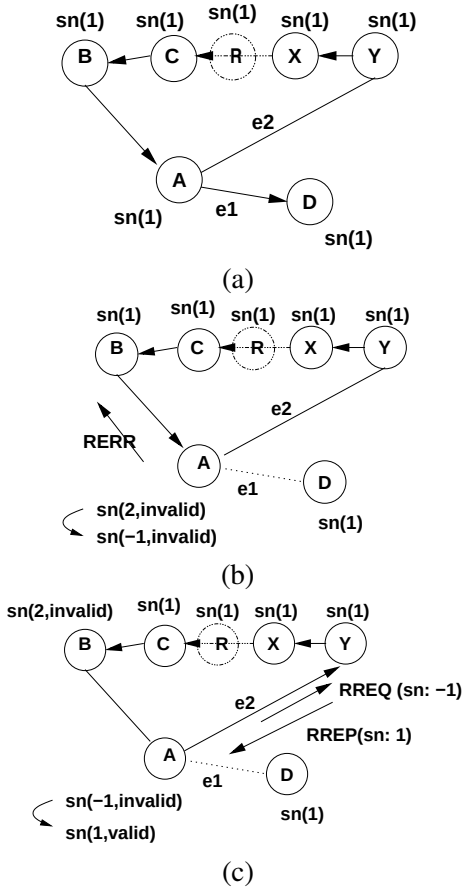


Figure 2.2: AODV count-to-infinity example

cause counting to infinity for destination D. Assume that link $e1$ fails, which results in node D being isolated from the connected component consisting of nodes A,B,C,$\{R\}$,X and Y. Now A detects the unreachability of D within a finite time through either a link-layer notification or HELLO messages. Node A invalidates the route to D, increments its sequence number for destination D to sn_D^2 (hence, $sn_D^2 > sn_D^1$), and sends a route error (RERR) to node B, which may not be delivered. This sequence of events is represented in Figure 2.2(b).

After the above sequence of events, node A searches for a new route to D with sn_D^2

as the required sequence number. However, destination D is unreachable and none of the other nodes can satisfy the request, because their sequence numbers equal sn_D^1. In our scenario, node B eventually invalidates its route entry for destination D, given that node A sends RERRs to B every time it receives a data packet for the invalid route to D.

Let t_1 be the time after which node A receives no data packets from B, node A deletes the invalid route for D with sn_D^2 at time $t_{del_D^A} = t_1 + DELETE\_PERIOD$. Node B invalidates its route entry for destination D at a time $t > t_1$, and notifies C of the unreachability of D proceeding in the same fashion as the RERR exchange between A and B. This results in C invalidating its routing entry for D. The nodes along path P learn about the unreachability of node D as the RERRs are propagated.

Let t_y be the time when Y invalidates its routing entry for D. At any time $t' < t_y$, any route search for destination D with a sequence number $sn_D > sn_D^1$ cannot be answered by any node in the network.

At a time $t_{req_D^A} > t_{del_D^A}$, node A sends a RREQ with an invalid sequence number for D. If at time $t_{rep_D^Y} < t_y$ node Y receives the RREQ, then it sends a RREP for D with a sequence number sn_D^1, which can now be used by node A to create a routing entry for destination D with sn_D^1. This results in the formation of an undirected cycle, which is shown in Figure 2.2(c). Similarly, once the $DELETE\_PERIOD$ elapses, node B deletes its invalid entry for D after attempting to find a route with sequence number sn_D^2. A RREQ for D by node B with an invalid sequence number can then be answered by node A with sn_D^1. This effect cascades to other nodes along the path, and counting-to-infinity occurs in this connected component. The route lifetimes at each node can be kept alive by the constant flow of data packets for D that

either originate locally at the node or are forwarded along the undirected cycle.

The basic problem can be summarized as follows: *A node A along a successor path P to destination D should never delete its invalid route table entry for D before guaranteeing that all its upstream nodes along path P have invalidated their active route entries for D.*

We note that a similar counting-to-infinity scenario can occur when nodes reboot after failures. On reboot, nodes running AODV wait for $DELETE\_PERIOD$ to elapse before engaging in routing operations; however, they forget their last-known sequence number for a destination. Hence, a counting-to-infinity scenario can be constructed that involves having all nodes along an upstream path rebooting.

In practice, counting-to-infinity and looping in AODV can be avoided by waiting "long enough" before deleting invalid routes or rejoining normal operation after reboots. However, as the network size and its diameter change, what "long enough" means must also change. Given that internodal coordination spanning multiple hops [7] incurs too much overhead and that very long waiting periods are undesirable for protocol efficiency, a more elegant solution to the counting-to-infinity and looping problems in the presence of routing-state loss is desirable, which we present in the next section.

2.3 Loop-Freedom using Sequence numbers

2.3.1 Sufficient conditions

We assume that node A maintains a strictly increasing sequence number (sn_A^A) for itself, and a sequence number sn_D^A and route cost d_D^A for a known destination node (D) in the

network. The last known sequence number and distance reported by neighbor B for destination D is stored by A as sn^A_{DB} and d^A_{DB}, respectively. Let $d^{A\star}_D$ be the smallest value of d^A_D assigned for a sequence number sn^A_D at node A. The link cost from node A to neighbor B is denoted by lc^A_B.

Sequence Number Condition (SNC): Node A can make node B its successor for destination D after processing an input event if $(sn^A_D < sn^A_{DB})$ or $(sn^A_D = sn^A_{DB} \ \wedge \ d^{A\star}_D \geq d^A_{DB} + lc^A_B)$. If no neighbor satisfies one of the above conditions, then node A must keep its current successor if it has any.

The proof that SNC can be used to enforce loop freedom is presented in [25] under the implicit assumption that nodes never "forget" the last sequence numbers they learn for a given destination and that no node ever forgets a sequence number that it maintains for itself, even after reboots.

2.3.2 Reseting destination sequence numbers

We present a new technique for nodes that delete or lose their sequence numbers for a destination to re-learn the current destination sequence number safely. We make the generic assumption that a node issues a query carrying an identifier (ID), and a reply to the query is issued by the destination carrying the same identifier (ID) after incrementing its own sequence number.

The following additional state is maintained at a node A: (i) A strictly increasing identifier (ID_A) used to identify queries that are used for reseting destination sequence numbers, and (ii) the first issued query identifier (FID^A_D) for a destination D at a time after which

the sequence number for that destination is no longer known. The FID_D^A for destination D is set to ∞ if no queries have been sent for this destination after the last reboot or loss of the destination sequence number; otherwise, it is set to the ID of the first issued query.

On a reboot, node A sets ID_A higher than the last used ID_A, and for every destination FID_D^A is set to ∞. Note that the variable ID_A is of local significance. The only requirement is that it be strictly increasing even after reboots, and can be derived from a real-time clock.

Sequence Number Reset Condition (SRC): If node A deletes or loses its previously known highest destination sequence number for D, it can safely update its routing-table entry for D with a reply issued by node D carrying an identifier ID, if $ID \geq FID_D^A$.

Theorem 1. *If a node uses SRC to obtain the sequence number for a destination, it is true that the sequence number stored at the node for that destination is strictly increasing with time.*

Proof. Assume that t_r is the last time that node A lost sequence number state for destination D, for which $sn_D^A(t_r) \leq sn_D^D(t_r)$. We now show that when node A assigns itself a new sequence number for destination D at any time $t > t_r$, it is true that $sn_D^A(t) > sn_D^A(t_r)$. At time $t_0 > t_r$ when node A participates, it is true that $ID_A(t_0) > ID_A(t_r)$, and $FID_D^A(t_0) = ID_A(t_0)$. To apply SRC at time t, node A must have received a reply issued at time t_2 carrying ID, in response to a query issued at a time $t_1 \geq t_0$, such that $ID = ID_A(t_1) \geq ID_A(t_0) \geq FID_D^A(t_0)$; where $t_1 < t_2 < t$. Because the destination must have incremented sn_D^D in the reply issued at time t_2 in response to query ID, it must be true that $sn_D^D(t_2) > sn_D^D(t_r)$. Hence, replies processed satisfying SRC will always ensure that the destination sequence number stored for a node is strictly increasing with time. □

2.4 On-demand Routing Framework

We use the same terminology introduced in Section 2.3 to present the on-demand routing framework based on destination sequence numbers.

2.4.1 Control Messages

We assume a generic control message framework consisting of route request (RREQ), route reply (RREP), and route error (RERR) messages similar to that of previous on-demand routing protocols.

A RREQ consists of the tuple $\{dst,\ src,\ rreqid,\ (src_1,\ rreqid_1),\ (src_2,\ rreqid_2),\ ...,\ (src_n,\ rreqid_n),\ msn_{dst},\ flags\}$. The field src denotes the identifier of the source that is seeking a path to the destination (dst), $rreqid$ is a request identifier that along with the source address (src) represents an unique RREQ generated by the source for destination dst. The $rreqid$ is set from the current ID_A stored at the node. The set of pairs ($src_i,\ rreqid_i$), where $i \in \{1, 2, ..., n\}$ are appended to the RREQ by relaying nodes (src_1, src_2, ..., src_n) that have no valid sequence-number for destination dst, and must re-learn it safely when they are part of this computation. msn_{dst} is the maximum of the destination sequence numbers along the path traversed by this RREQ, and $flags$ carries control bits. One control bit used is the *R*-bit that is set when the RREQ must be answered only by the destination.

A RREP consists of the tuple $\{dst,\ (src,\ rreqid),\ (src_1,\ rreqid_1),\ (src_2,\ rreqid_2),\ ...,\ (src_n,\ rreqid_n),\ sn_{dst},\ d_{dst},\ ttl,\ flags\}$. The field ttl states the lifetime of the route at the node relaying the RREP, $rreqid$ is carried in the RREP to forward it along the reverse path

to the source using information cached for the RREQ which is uniquely identified by the pair $(src,\ \ rreqid)$. The set of pairs $(src_i,\ rreqid_i)$, where $i \in \{1, ..., n\}$, are copied from the RREQ when the RREP is generated by the destination. It is necessary to carry the pairs for all the relaying nodes because SRC needs to be applied individually at each node to do a safe sequence-number reset for the destination. sn_{dst} is the destination sequence number stored at the relaying hop, d_{dst} is the distance to the destination at the relaying hop, and $flags$ contains the *'D'* bit, which may be set if the destination originates the RREP.

The RERR is the tuple $\{orig,\ unreachdests\}$, where $orig$ denotes the node originating the route errors, and $unreachdests$ is the list of destinations that are not reachable at $orig$.

2.4.2 Information Stored

Node A maintains a strictly increasing request identifier ID_A for the issuing of RREQs, and a sequence number for itself sn_A^A. The routing-table entry at node A for destination D includes the current sequence number (sn_D^A), the current route cost (d_D^A), the successor (s_D^A), and the state of the route (rt_D^A), which can be valid or invalid. If no routing entry exists at node A for destination D, then the current sequence number is considered invalid (i.e., $sn_D^A = -1$), and cannot be used to check if loop-free conditions are satisfied. For unknown routing entries, a first used request identifier (FID_D^A) is stored to keep track of the $rreqid$ of the RREQ that was issued after the last reboot or state loss. The FID_D^A is set to ∞ after losing the sequence number for destination D or after a reboot. A cache maintains a list of RREQs identified by their $(source, rreqid)$ pairs. For each cached pair, a corresponding previous hop-address is stored.

2.4.3 Generation of Sequence Numbers

Nodes must generate their destination-sequence numbers from a 64-bit real-time clock. This method was proposed in LDR [8], and is necessary to prevent loops that can be caused when a destination reboots and cycles to a previously used sequence number. The request identifiers (ID_A) must be similarly derived from a real-time clock. This is essential because the node loses its state for the last used request identifier when it reboots and participates immediately in the routing process for a destination. If previously used request identifiers are repeated in the RREQs, they are not forwarded and no RREQs reach the destination. We note that this scheme should be applied to other 'soft-state' protocols like the Dynamic Source Routing (DSR) [13] protocol or any on-demand routing protocol that performs route searches using the $(source, rreqid)$ pair.

2.4.4 Conditions

Based on the previously defined state information stored at each node and the control messaging, we define the following conditions for on-demand routing. The superscripts req and rep are used for variables included in a RREQ and RREP, respectively.

ASC: (Accept Sequence-number Condition). When node A receives a RREP from node B for destination D, then node A sets $s_D^A \leftarrow B$ if $(sn_D^A < sn_D^{rep})$ or $(sn_D^A = sn_D^{rep}) \ \wedge \ (d_D^A \geq d_D^{rep} + lc_B^A)$. If $sn_D^A = -1$, then node A should accept a RREP only if there exists a pair $(src, rreqid) \in$ RREP, such that $src = A \wedge rreqid \geq FID_D^A$, and $D_D^{rep} = 1$ (i.e., generated by the destination).

*S*SC: (Start Sequence-number Condition). Node I can issue a RREP responding to a RREQ for destination D in the following two cases: (a) $I \neq D$, I has an active route to D, $sn_D^I > sn_D^{req}$, and $R_D^{req} = 0$, or (b) $I = D$. For case (b), if $msn_D^{req} = sn_D^D$ or $R_D^{req} = 1$, node D increments sn_D^D by one. If $R_D^{req} = 1$, it must set $D_D^{rep} = 1$.

*M*SC: (Maximum Sequence-number Condition). If node A relays a RREP for destination D, it sets $sn_D^{rep} \leftarrow sn_D^A$. The relayed RREP must not change the value of D_D^{rep}. Node A relays a RREQ for destination D only if A has not previously processed this RREQ and sets $msn_D^{req} = max\{msn_D^{req},\ sn_D^A\}$. If $sn_D^A = -1$ or $R_D^{req} = 1$, then it sets $R_D^{req} = 1$ in the relayed request.

*U*SC: (Update Sequence-number Condition). If node A must change s_D^A, it sets $d_D^A \leftarrow \infty$, increments ID_A, and sends a RREQ carrying msn_D^{req}=sn_D^A.

*R*SC: (Reset Sequence number Condition). If node A has no route entry for D (i.e., $sn_D^A = -1$) then it must always set $R_D^{req} = 1$ in any RREQ originated or relayed. When relaying a RREQ, the node increments ID_A and appends the pair $(src = A,\ rreqid = ID_A)$ to the RREQ. If FID_D^A is ∞, then it sets $FID_D^A \leftarrow ID_A$.

2.4.5 Basic Route Operations

2.4.5.1 Node states

For a destination, a node is said to be *active* for a computation (A, ID_A) if it is the source of a RREQ identified by $(src = A,\ rreqid = ID_A)$. A node is *engaged* in a RREQ computation (A, ID_A) by caching the corresponding previous hop address. Otherwise, the

node is *passive*.

At any given time, a node can be active for at most one RREQ for the same destination. The RREQ (A, ID_A) terminates when either node A attains an acceptable route reply for destination D or the timer for its RREQ expires. A node may be engaged in multiple RREQs for the same destination, but relays a RREQ (A, ID_A) only once by caching the pair it forwards.

2.4.6 Message Handling

2.4.6.1 Route Requests

Node A becomes active for a RREQ (A, ID_A) when it has data packets to send for destination D, but no active valid route entry. RREQ's are issued as per RSC and USC. RREQ's issued by the source can have increasing ttl's controlled by the IP header or by additional means to support expanding ring searches. For every active RREQ computation, a timer is setup as follows: *RREQ timer* $\leftarrow$ *(*2.ttl.latency) (where ttl is the time-to-live of the broadcast flood and latency is the estimated per-hop latency of the network). If node A receives no RREP after the expiry of its timer for RREQ (A, ID_A) for destination D, it sends a new RREQ with an increased ttl. If node A does not receive a RREP for destination D after a number of attempts, a failure is reported to the upper layer.

When node B relays a RREQ (A, ID_A), it caches the pair (A, ID_A) and the corresponding previous hop (A). The RREQ is relayed in accordance with MSC and RSC. Node B is then engaged in this computation (A, ID_A).

2.4.6.2 Route Replies

A RREP is generated in response to a RREQ (A, ID_A) by node I (which can be D) if SSC is satisfied.

RREPs that satisfy ASC are processed as per Section 2.4.6.3. The RREP identified by (A, ID_A) is relayed along the reverse path using cached entries for (A, ID_A) and follows MSC.

2.4.6.3 Updating Routing Tables

After node A accepts a RREP rep for destination D from neighbor B that satisfies ASC, node A updates its routing-table entry for D as follows: $s_D^A \leftarrow B$, $sn_D^A \leftarrow sn_D^{rep}$, and $d_D^A \leftarrow d_D^{rep} + lc_B^A$. The ttl for a route entry is set to ttl_D^{rep}, and the RREP is relayed with the time to live for the destination.

2.4.6.4 Routing Table Maintenance and Route Errors

Node A invalidates a route entry for destination D with S as the next hop, in one of the following ways: (i) No data packet is forwarded using this route entry for *active_route_timeout* seconds (the time after which a route-entry expires); (ii) A link-failure notification for the next hop S is received; or (iii) A RERR is received, which indicates that D is no longer reachable through S. A node A invalidating an entry performs the following steps: It sets $rt_D^A = invalid$, $s_D^A \leftarrow \phi$, and $d_D^A \leftarrow \infty$. A route entry with state $rt_D^A = invalid$ can be purged at any time to save memory. For cases (ii) and (iii), after determining the set of destinations affected by this event, node A sends a RERR to all the predecessors (either as a broadcast or separate unicasts).

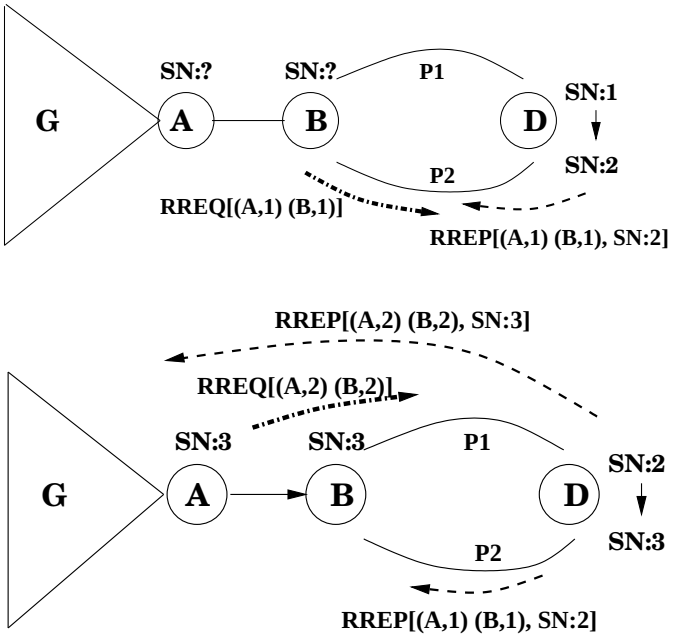


Figure 2.3: Re-learning sequence numbers safely after losing state

2.4.6.5 Routing Table Maintenance

A node can delete its routing table entry for a destination at any time, and is not required to store the destination sequence number for arbitrary periods of time to ensure correct protocol operation.

2.4.6.6 Reboots

On a reboot, a node can participate in the routing protocol activities without having to wait for any arbitrary periods of time. A node sends RERRs in response to data packets received and intended for nodes other than itself, until routes are re-established.

2.4.7 Example

Figure 2.3 shows the following sequence of events in a network when nodes lose state routing information about a node and attempt to re-learn it using the rules of the framework.

Assume that at time t, nodes A and B, and the nodes in graph G form a part of the DASG for destination D and all of them have stored a sequence number of one for D. At time $t_1 > t$, nodes A and B lose their route entry state (denoted by SN:? in the figure) for D, and node A issues a RREQ with a pair $(A, 1)$ and the *R*-bit set. Node B relays the RREQ after appending $(B, 1)$ to the list. Node A and B set their respective FID_D^A and FID_D^B to 1. Assume that the RREQ traversing path $P2$ reaches D. Node D initiates a RREP with an increased $sn_D^D = 2$, *D*-bit set, and the list of pairs $[(A, 1), (B, 1)]$. Let node A's RREQ timer expire because the RREP traversing the reverse path $P2$ is delayed, and node A issues a new RREQ $(A, 2)$ which is relayed by B with $(B, 2)$. Assume that this RREQ traverses the path $P1$ and reaches D. A RREP is generated by D with an increased $sn_D^D = 3$, *D*-bit set and the pairs $[(A, 2)(B, 2)]$. The RREP is received by nodes B and A, and they update their routing table entries for D because SRC is satisfied and set sn_D^A, and sn_D^B to 3. If node B loses its state again at a time $t_2 > t_1$, it will not process the old delayed RREP with the *D*-bit set along path $P2$. This is because node B sets FID_D^B to ∞ after losing state, or if node B issued a new RREQ at some time greater than t_2 then $FID_D^B > 2$, and the RREP carrying the pair $(B, 1)$ cannot be accepted. Hence, node B avoids re-learning old sequence numbers for destination D from stale RREPs, and the ordering of sequence numbers along the DASG for D is strictly maintained at all times.

2.4.8 Analysis

Theorem 2. *If nodes use the rules of the framework, then they always maintain a strictly increasing sequence number for a destination even after state loss.*

Proof. The conditions for the framework follow the same rules as discussed in section 2.3.2. The RREQ with *R*-bit set serves as a query and the RREP with *D*-bit set serves as the reply tagged by the destination. For a node A, the strictly increasing request identifier ID_A is used for tagging the RREQs and is carried in the RREPs generated by the destination. A FID_D^A is maintained to keep track of the identifier of the first issued RREQ. ASC encompasses the SRC condition when processing RREPs for a destination with no entry. A RREP relayed from the destination will always carry the same or a higher destination sequence number. Hence, from the same argument as in Theorem 1, this proof is direct. □

Theorem 3. *The framework ensures that if a path* P=$\{n_k, ..., n_1\}$ *exists at some point in time as defined by the successor entries of the nodes along the path, it is true that* $sn_{n_1}^{n_i} < sn_{n_1}^{n_{i-1}}$ *or* $sn_{n_1}^{n_i} = sn_{n_1}^{n_{i-1}} \wedge d_{n_1}^{n_i} > d_{n_1}^{n_{i-1}}$, *for* $i \in \{2, k\}$.

Proof. For path P to exist at a given time t, it must be true that all nodes in P have a valid successor. Node n_i can make n_{i-1} its successor when it receives a RREP from n_{i-1}, carrying $sn_{n_1}^{rep} = sn_{n_1}^{n_{i-1}}$ and $d_{n_1}^{rep} = d_{n_1}^{n_{i-1}}$. There are two possible cases of ASC: Case (i), node n_i has a valid $sn_{n_1}^{n_i}$ and must follow SNC to accept the RREP. It must be true that $sn_{n_1}^{n_i} < sn_{n_1}^{rep}$, or $sn_{n_1}^{rep} = sn_{n_1}^{n_i} \wedge d_{n_1}^{rep} + lc_{n_{i-1}}^{n_i} \leq d_{n_1}^{n_i}$. After updating the route entry, it is still true that $sn_{n_1}^{n_i} = sn_{n_1}^{n_{i-1}} \wedge d_{n_1}^{n_i} > d_{n_1}^{n_{i-1}}$. This cannot affect a predecessor n_{i+1} because $sn_{n_1}^{n_i}$ only increases or remains the same with decreased $d_{n_1}^{n_i}$. Case (ii), node n_i has no valid $sn_{n_1}^{n_i}$ and it uses SRC to accept the

RREP. After node n_i updates its routing table, it is direct that $sn_{n_1}^{n_i} = sn_{n_1}^{rep} \wedge d_{n_1}^{n_i} > d_{n_1}^{rep}$, because link costs are greater than zero. Also, from Theorem 2, node n_i must have a higher $sn_{n_1}^{n_i}$ than it previously reported in its RREPs before time t to a predecessor n_{i+1}, and at time t, it must be true that $sn_{n_1}^{n_{i+1}} < sn_{n_1}^{n_i}$. Hence, if path P exists at time t, every node n_i, where $i \in \{2, k\}$ must have accepted a RREP following one of the two cases, and therefore, the theorem is true. □

Theorem 4. *At any instant, the framework is loop-free.*

Proof. This proof is direct from the proof of SNC [25] because Theorem 3 shows that the framework maintains an ordering of fresher sequence numbers or equal sequence numbers with decreasing distances along any successor path to the destination. □

Theorem 5. *In a connected component, all nodes partitioned from a destination will invalidate their routing entries for that destination within a finite time*

Proof. Consider a connected component G in which all nodes are partitioned from destination D. If any node $n \in G$ has an active route for D, it must be true that a DASG for destination D must exist in G. Assume that at time t, the destination D is partitioned from the set of nodes $n \in G$. By default RERR propogation, all nodes in the DASG should receive a RERR in finite time, say $t_{term} > t$ and invalidate their routing entries. We have to prove that a node never re-learns a route from its upstream nodes in the DASG in the presence of link failures and node state loss (which will cause cyclic propogation of RERRs). Let $t_{last} > t$ be the time when destination D initiated the final RREP with *D*-bit set. At a time $t_{proc} > t_{last}$, all RREPs initiated with a *D*-bit must have been processed at the sources that issued the corresponding RREQs. At any time later than t_{proc}, there are two cases by which any node $n \in G$ can learn

a route for destination D: Case (i), a RREP with a *D*-bit set, which is however not possible as the destination is partitioned and all old RREPs have been processed. Case (ii), node n can receive a RREP from a neighbor $A \in G$ satisfying SNC. However, neighbor A cannot be upstream of n in the DASG, because it is loop-free at every instant (Theorem 4). Hence, nodes can only switch to successor nodes that are downstream in the DASG. So, assuming finite message time, the default propogation of RERR messages will reach all nodes in the DASG without going through a cycle, and all nodes will invalidate their routing entries a finite time after $t_{term} > t_{proc} > t$. □

Theorem 6. *In an error-free stable connected network, a source will establish a route to a destination in finite time.*

Proof. This proof of convergence is similar to the one for LDR [8] (pp.60, Theorem 5) considering just the cases with sequence numbers. A RREQ traverses a path P, and carries a msn that will be the highest of all the destination sequence numbers stored at the nodes. A RREP can only be generated by a node storing a higher sequence number or the destination itself. Hence, the RREP must satisfy ASC at all the nodes which relay the RREP along the reverse path. If any node along the path has a higher sequence number than the one carried in the RREP, then it must have learned a better route, but can still relay the RREP. A node relaying a RREQ with the *R*-bit set is equivalent to relaying the RREQ with a msn set to the highest known destination sequence number in the network (i.e., the one stored at the destination). The rest of the details are identical because a RREQ with *R*-bit set generates a RREP with the highest sequence number and *D*-bit set which will be acceptable at all nodes relaying the RREP along the reverse

path. □

2.5 AODV with Robust Sequence Numbering

We present a modification to the way in which destination sequence numbers are managed in AODV as an example of our framework, and call it AODV-RSN (for robust sequence numbering). In a nutshell, in AODV-RSN, the time period $DELETE\_PERIOD$ is eliminated and a routing-table entry can be deleted without the necessity to wait for any arbitrary time period. Nodes can participate in routing actions immediately after a reboot. We note the necessary changes in the control signalling and the information maintained at each node with respect to AODV [19].

2.5.1 Control Message modifications

The following changes to the AODV specification are required for realizing AODV-RSN. We utilize the 'unknown sequence number' *U*-bit of the RREQ, which is equivalent to the *R*-bit in the framework specification. The RREP requires the addition of a new 'Destination Initiated Reply' *D*-bit (which is borrowed from the reserved 13-bits), and a 64-bit field for carrying the request identifier $rreqid$. The parameters stored in the routing table do not require any modifications. The additional (source, rreqid) pairs in both RREQs and RREPs must be carried in extra headers.

2.5.2 Stored Information

For each RREQ that a node processes, it caches the (source, rreqid) pair and the address of the neighbor that transmitted the RREQ. The node stores a *First Issued Identifier (FID)* for every known destination without a routing-table entry.

2.5.3 RREPs and RREQs

RSC and USC must be followed when RREQs are issued or relayed and the information relayed in RREQs and RREPs must adhere to MSC.

We note that the AODV specification requires that nodes invalidate their routing table entries by incrementing the corresponding destination sequence number. Intermediate RREPs are generated if the RREQ carries a sequence number less than that stored at the node. This differs from our framework specification, and in AODV-RSN only the destination can alter its sequence number. This is because of the modified routing table maintenance and the SSC. A RREQ must carry the pair $(src, rreqid)$ and the pairs $(src_i, rreqid_i)$, where $i \in \{1, ..., n\}$, of relay nodes that have no routing entries for the corresponding destination. The RREP generated in response to a RREQ by the destination must carry the entire set of (source, rreqid) pairs along with the *D*-bit set, and must be relayed hop-by-hop with this information unchanged.

The reverse hop cached for the (source, rreqid) pair and the relaying rules are necessary to relay the RREP along the same reverse path. This is a necessary condition for the convergence of the protocol. If the more recent specification for AODV [30] is used, then the reverse hop caching rules for relaying RREQs and RREPs are not required because the RREQs and RREPs carry the path traversed.

2.5.4 Reverse Routes

A RREQ carrying the sequence number for the source can be considered as equivalent to a RREP for the source. However, to satisfy ASC, such a RREQ can only be accepted if a node has previous knowledge of the destination sequence number for this source. Otherwise, it has to initiate a RREQ using the standard mechanism to re-learn the sequence number of the source.

2.6 Sequence Number Window Routing Protocol

2.6.1 Sufficient conditions for loop-freedom using sequence number labels

In the past, loop-free routing approaches based on destination sequence numbers have relied on the premise that a sequence number serves the purpose of time-stamping an update, and thus accepting the update for a destination with the latest sequence number maintains loop-freedom. This is the approach (based on SNC) that has been used in the past for loop-free routing based on destination sequence numbers ([19], [25], [29]). Mosko and Garcia-Luna-Aceves [27] propose the use of label sets that allows nodes to re-use the last known sequence number. However, when no new unique labels can be found from the set, a new updated sequence number has to be requested from the destination.

Rather than considering sequence numbers as time-stamps, we model the sequence numbers of a destination as a finite label space from $[0, \ldots, 2^{n-1}]$ in which n is the number of bits allocated for storing the sequence number. By doing so, the problem of maintaining loop-freedom using sequence numbers reduces to a case of assigning destination sequence numbers

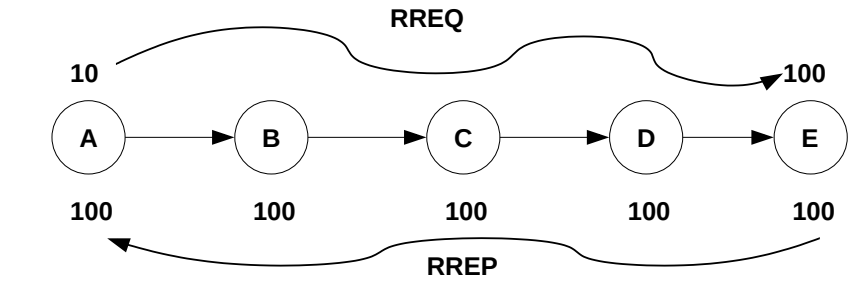
as labels for a destination at each node along the successor path. Following this approach, we augment SNC with the following sufficient condition, which allows nodes to label themselves with a sequence number for a destination without creating loops.

Sequence Label Condition (SLC): Let Rsn_D^A denote the sequence number for destination D that was last reported by node A in an update to its neighbors. When A makes B its successor for destination D after processing an input event that satisfies $sn_D^A < sn_{DB}^A$, node A can assign (label) itself a destination sequence number that satisfies $Rsn_D^A < \ sn_D^A \leq \ sn_{DB}^A$.

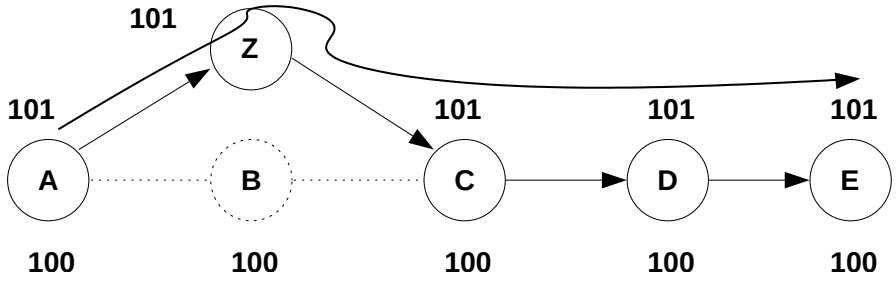
Theorem 7. *Using SNC for choosing successors and SLC for updating sequence numbers at nodes cannot create loops if no node ever forgets the largest sequence number it learns for a destination.*

Proof. The proof follows from the fact that SNC is sufficient to enforce loop freedom if $sn_{n_1}^{n_i} < sn_{n_1}^{n_{i-1}}$, or $sn_{n_1}^{n_i} = sn_{n_1}^{n_{i-1}} \wedge \ d_{n_1}^{n_i} > d_{n_1}^{n_{i-1}}$, for $i \in \{2, k\}$ along any successor path P = $\{n_k, \ldots, n_1\}$ to destination n_1.

For path P to exist at a given time t, it must be true that all nodes in P have a successor. Node n_i applies SLC before making n_{i-1} its successor if $sn_{n_1}^{n_i} < sn_{n_1 n_{i-1}}^{n_i}$, and it updates its route entry for n_1 to one of these cases: (i) $sn_{n_1}^{n_i} < sn_{n_1 n_{i-1}}^{n_i}$, or (ii) $sn_{n_1}^{n_i} = sn_{n_1 n_{i-1}}^{n_i}$ $\wedge \ d_{n_1}^{n_i} > d_{n_1 n_{i-1}}^{n_i}$. At any later time, $sn_{n_1}^{n_{i-1}}$ can only increase or remain the same with non-increasing $d_{n_1}^{n_{i-1}}$. In both cases, $sn_{n_1}^{n_i} > Rsn_{n_1}^{n_i}$. If a neighbor node n_{i+1} is using node n_i as its successor or applies SLC to update its route entry before switching to node n_i, it can only set itself a label $sn_{n_1}^{n_{i+1}} \leq sn_{n_1 n_i}^{n_{i+1}} \leq Rsn_{n_1}^{n_i}$. Therefore, it is true when node n_i applies SLC that $sn_{n_1}^{n_{i+1}} < sn_{n_1}^{n_i}$, and the ordering along path P is maintained. If SNC is used, and node



(a) AODV - Sequence numbers as absolute time stamps



(b) AODV - On Route Failure

Figure 2.4: Sequence number assignment in AODV

n_i makes n_{i-1} as successor when $sn_{n_1}^{n_i}{=}sn_{n_1}^{n_{i-1}}$ and $d_{n_1}^{n_i}{\geq}d_{n_1 n_{i-1}}^{n_i}{+}lc_{n_{i-1}}^{n_i}$, the ordering is still maintained. Hence, the theorem is true. □

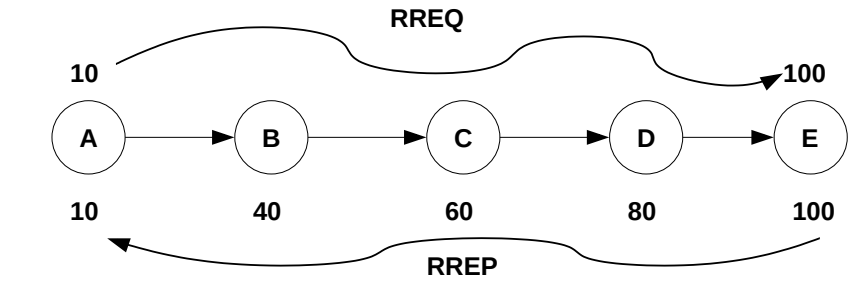
2.6.2 Sequence Number Windows

SLC allows nodes to assign themselves a destination sequence number *smaller* than the latest destination sequence number received in an update. The only requirement is that the node chooses a new sequence number that is greater than the last sequence number it reported to its neighbor nodes for this destination.

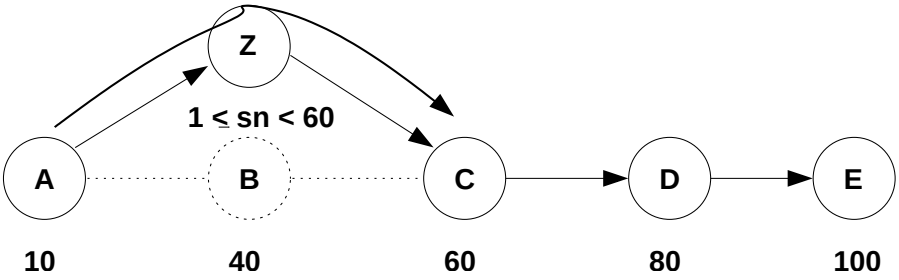
In past approaches like AODV, a node updates itself to the latest destination sequence number reported in a control message. However, this results in the destination node being the only one that can resolve most RREQs. For example, Figure 2.4 (a) shows a five-node network

topology. Node A has an invalid route to E with a sequence number $sn_D^E = 10$. Destination E has a sequence number $sn_E^E = 100$. Node A trying to establish a route to E sends a RREQ that is answered by the destination E, and the RREP carries $sn_E^{rep} = 100$. Nodes B,C, and D along A's successor path update their destination sequence numbers for E to 100. Now, node B fails and an alternate path through node Z exists at a later time as shown in Figure 2.4 (b). Node A attempts to re-establish its route to E. Node A's RREQ carries a increased sequence number $sn_E^{req} = 101$, which prevents any of the nodes along the path ZCD from replying, even though nodes C and D have a valid active route. Eventually, the RREQ reaches the destination E, which now responds with a increased sequence number $sn_E^E = 101$ and the successor path ZCD from A to E is established.

By allowing a *progressive* ordering of the sequence numbers, it is possible to expedite route recovery and reduce control overhead. We introduce the *Sequence Number Window* (SNW) as a tool to achieve this type of ordering. Figure 2.5 (b) illustrates a sequence number window between node A and E in the interval $[10, ..., 100]$ assuming an initial configuration where nodes B,C, and D have no routes for E. Nodes label themselves with sequence numbers smaller than the latest known sequence number, which amounts to distributing the sequence numbers inside the window while maintaining the ordering. Figure 2.5 (b) illustrates the benefits of distributing sequence numbers inside a window compared to AODV's approach. As in the previous example, node B fails, and node A obtains a reply from node C when it attempts to obtain a new path to E, because A's increased sequence number in the request $sn_E^{req} = 11$ can be satisfied by node C. Node Z is assumed to have a valid sequence number in the range $1 \leq sn_E^Z < 60$. With on-demand routing protocols that resort to expanding ring searches, this



(a) Window of Sequence numbers



(b) Fast recovery on route failure

Figure 2.5: Sequence number windows

scheme enables more nodes with active routes to E to respond.

2.6.3 Conditions

SWR augments the rules of our basic framework with a condition for detecting the boundary of windows.

*W*BC: (Window Boundary Condition). If node A receives a RREQ for destination D and $sn_D^A > msn_D^{req}$ then node A sets the window count wc to 1, otherwise it increments wc by 1.

WBC allows nodes relaying a RREQ to determine distributedly the start and end of sequence number windows, and the number of hops spanned by the window. A window count

of one indicates the beginning of a new window and signals the end of a previous sequence number window (except at the source, where the window can only begin). The window count indicates the number of nodes over which the current sequence number window is being built.

2.6.4 Additional Information and Control Messages

A node that processes a RREQ must cache the RREQ identifier given by (source, rreqid), and an additional tuple consisting of $(msn, wc, reset)$. The parameter msn is copied from the RREQ, wc is the window count used to keep state for creating sequence number windows in a distributed fashion, and $reset$ is used to decide when updating routing tables whether the node must update to the sequence number carried in the reply. The $reset$ copied from the *R*-bit of the RREQ allows nodes to determine locally whether they are part of a sequence number window being constructed, or if they have to update themselves to the highest sequence number to maintain the ordering along the path being created. At a node B that has cached a RREQ (A, ID_A), we refer to these entries as $msn^B_{(A,ID_A)}$, etc,.

RREQs carry an additional window count, wc, which is used for identifying the number of hops over which the distributed sequence number windows extend. RREPs and RERRs do not require any changes.

2.6.5 Message Handling

2.6.5.1 Initiating RREQs

When node A initiates a RREQ req on a computation (A, ID_A) for destination D, it sets $wc^{req}_D \leftarrow 1$.

2.6.5.2 Relaying RREQs

A node relaying a RREQ calculates wc following WBC. The new value of wc, msn_D^{req}, along with $reset \leftarrow R_D^{req}$ are cached for the RREQ computation.

2.6.5.3 Initiating RREPs

If the destination D receives a RREQ req such that $R_D^{req} = 0$, and $msn_D^{req} = sn_D^D$, it increments sn_D^D by a parameter $dstSeqInc$ before issuing the RREP.

A linear-increment scheme with a pre-configured $dstSeqInc$ parameter should suffice for most network configurations. However, performance can be improved by using adaptive-increment schemes that derive $dstSeqInc$ as a function of the prevailing network conditions (i.e., number of RREQs received within a time interval).

2.6.5.4 Updating Route Entries

Node A updates its routing information when it accepts a RREP $\{D, S, ID_S, sn_D^{rep}, ttl, d_D^{rep}, D\}$ from neighbor B. Node A calculates sn_D^{adj} after retrieving the values of $msn_{(S,ID_S)}^A$ and $wc_{(S,ID_S)}^A$ from the cache for the relayed RREQ (S, ID_S).

$$sn_D^{adj} = sn_D^{rep} - \lfloor \frac{sn_D^{rep} - msn_{(S,ID_S)}^A}{wc_{(S,ID_S)}^A + 1} \rfloor \tag{2.1}$$

Node A updates its routing table entry for D according to its current state (rt_D^A) as follows:

Case (a): If $rt_D^A = \phi \ \vee \ reset_{(S,ID_S)}^A = 1$, then

$$sn_D^A \leftarrow sn_D^{rep} \tag{2.2}$$

For cases (b), and (c), rt_D^A can be $valid$ or $invalid$, and $reset_{(S,ID_S)}^A$ must be 0.

Case (b): If $sn_D^{rep} > sn_D^A$, then

$$sn_D^A \leftarrow \begin{cases} sn_D^{adj} & \text{if } msn_{(S,ID_S)}^A \geq sn_D^A \\ sn_D^A + 1 & \text{otherwise} \end{cases} \tag{2.3}$$

Case (c): If $sn_D^A{=}sn_D^{rep}$, and $d_D^A{\geq}d_D^{rep} + lc_B^A$, then sn_D^A is not changed.

If the route reply was used to update the routing table sequence number entry, then node A sets $s_D^A \leftarrow B$, and $d_D^A \leftarrow d_D^{rep} + lc_B^A$.

2.6.6 Local Route Repair

To take advantage of the progressive ordering of sequence numbers as illustrated in Section 2.6.2, SWR uses a local repair mechanism to avoid informing the source of the failure of a route. When an intermediate node I experiences a link failure towards the next hop for a destination, it will send a RREQ as per USC setting $ttl \leftarrow 1$, and $RREQ\ expiry\ timer \leftarrow 2.ttl.latency$. This is equivalent to a one-hop neighbor query that allows it to recover routes locally. The node I must buffer data packets while waiting for a RREP. If no RREPs are received, then RERRs are sent as per the default mechanism (Section 2.4.6.4).

2.6.7 Reverse Routes

A RREQ generated by a source can be considered as a RREP in the reverse direction. However, SNWs cannot be used for setting up routes in the reverse direction, because of the lack

of window boundaries. Hence, SWR uses an optimization to use SNWs. If node A receives a RREQ from B and has no valid route towards the source S of the RREQ, node A creates a new route entry for the source S, sets $s_S^A \leftarrow B$, sets the lifetime of the route equal to the *reverse route lifetime*, and flags the route entry indicating that it is an invalid reverse route. When node A has a flagged reverse route to S, and needs to send data packets to that destination, it sends a unicast RREQ to s_S^A. This RREQ is forwarded on a hop-by-hop basis along a path of nodes with invalid reverse routes to S, and a RREP can be generated by either a node satisfying SSC or the destination. A unicast RREQ follows the same rules as a broadcast RREQ.

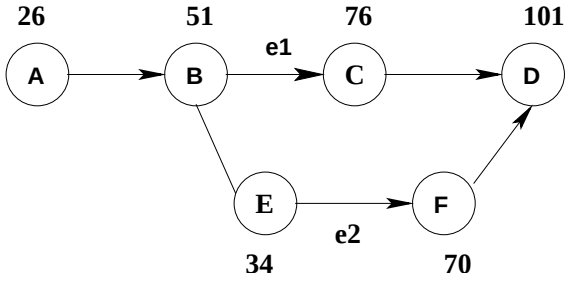
2.6.8 SWR Example

We illustrate an example of the creation of a sequence-number window. We show how progressively ordered sequence numbers allow more intermediate nodes to reply to RREQs, and the local repair mechanism to resolve a failure without notifying the source.

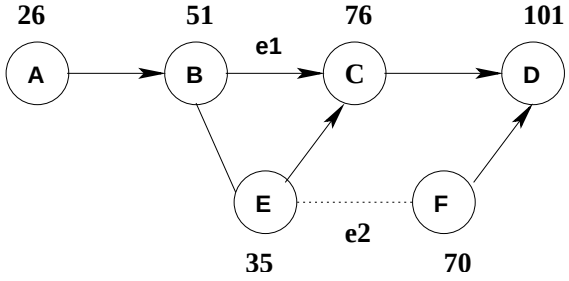
2.6.8.1 Sequence Number Window Creation

To illustrate the use of SNWs, assume that the nodes in Figure 2.6 (A) have an invalid route for destination D, and store a corresponding sequence number of 1.

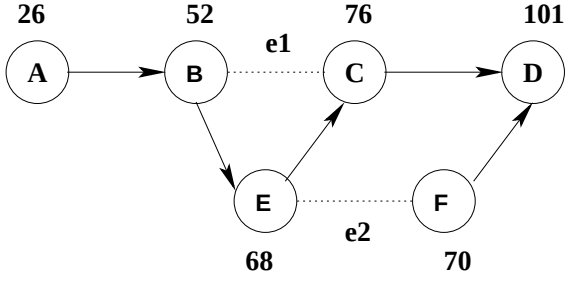
Now, assume node A starts a RREQ identified by (A, ID_A) carrying $(msn = 1,\ wc = 1)$ for destination D. Nodes B and C relay the RREQ after increasing the wc in the RREQ to two and three, respectively. Node D, on receiving the RREQ, increments its sequence number to 101 using a linear increment (say $dstSeqInc = 100$). It generates a RREP identified with $(A,\ \ ID_A)$ and carrying $sn = 101,$ and $d = 0$.



(a) Time t_1



(b) Time t_2



(c) Time t_3

Figure 2.6: SWR operation - Example

Node C accepts the RREP because it satisfies ASC, adds a new routing table entry for destination D, sets $s_D^C \leftarrow D$, and calculates a sequence number of 76 for destination D using Eq. 2.3 (i.e., $101 - \lfloor \frac{(101-1)}{3+1} \rfloor$). Similary, node B and node A update their routing tables using Eq. 2.3 to setup an entry for destination D with sequence numbers, 51 and 26, respectively. At time t_1, there is a progressive ordering of destination sequence numbers from node A to destination D. The RREQ and RREP are identified by their unique computation pair (A, ID_A),

and the corresponding values for the computation are cached are retrieved based on this pair.

After a similar sequence of events, assume that nodes E and F have set their destination sequence number for D to 34 and 70, respectively, at some time $t < t_1$.

2.6.8.2 Route Maintenance

At time $t_2 > t_1$, link e_2 fails. Node E detects a link failure and sends a RREQ with ($msn_D^{req} = 34,\ wc = 1$) which evokes a RREP with sn_D^{rep}=76 from C that satisfies SSC. Node E activates routing entry for D after processing the reply, and updates $sn_D^E \leftarrow 35$ as per Eq. 2.3. Figure 2.6(b) shows the state of the network.

At a later time $t_3 > t_2$, let node B detect the failure of link e_1 and sends a RREQ with ($msn_D^{req} = 51,\ wc = 1$). Node E does not satisfy SSC and relays the RREQ with ($msn_D^{req} = 51,\ wc = 2$). Note that there is a window between B and C spanning node E. Node C responds to the RREQ with a RREP carrying $sn_D^{rep} = 76$, because SSC is satisfied. Node E processes the RREP and sets $sn_D^E \leftarrow 68$ as per Eq. 2.3, which amounts to redistributing sequence numbers in the window between B and C. Node B re-establishes a route to D, and updates its route entry to set $sn_D^B \leftarrow 52$ and $s_D^B \leftarrow E$. In this case, we assume that node B has sent a RERR to node A which may perform its own simultaneous route search for D; however, for simplicity, we do not depict that. Figure 2.6(c) shows the DASG at time t_3.

2.6.8.3 Local Repair

We illustrate the intermediate one-hop local repair of SWR with the following sequence of events. At time $t_4 > t_3$, link e_1 comes up and link BE fails. Node B detects the

failure, and now attempts to perform a local repair (assuming it is not a source). Node B sends a RREQ with $msn = 52$, which is answered by node C with $sn = 76$. Node B switches successors to node C and recovers the route locally. The progressive ordering of sequence numbers allows intermediate nodes to repair routes by querying neighbors.

The same benefits can also be illustrated without local repair by considering the mobility of the network. If source A moves closer to C, and attempts to re-establish a route with an expanding ring search. Node A will receive a RREP from C in its first attempt. This avoids the necessity to flood all the way to the destination which happens to be the pre-dominant case in AODV.

2.6.9 Analysis

Theorem 8. *In SWR, if a path P=$\{n_k, ..., n_1\}$ exists at some point in time as defined by the successor entries of the nodes along the path, it is true that $sn_{n_1}^{n_i} < sn_{n_1}^{n_{i-1}}$ or $sn_{n_1}^{n_i} = sn_{n_1}^{n_{i-1}} \wedge d_{n_1}^{n_i} > d_{n_1}^{n_{i-1}}$, for $i \in \{2, k\}$.*

Proof. For path P to exist at a given time t, it must be true that all nodes in P have a successor. The route entry for destination n_1 at node n_i can be in one of the three states when it accepts a RREP (satisfying ASC) for n_1 identified by $(src, rreqid)$ from neighbor node n_{i-1}: (i) no information, (ii) invalid , or (iii) valid. The RREP carries $sn_{n_1}^{n_{i-1}}$, and $d_{n_1}^{n_{i-1}}$. Node n_i switches successors to node n_{i-1} and updates its route entry for n_1 in one of the following ways.

Case (i): The route entry does not exist ($sn_{n_1}^{n_i} = -1$), and node n_i must use SRC to process the reply, and the ordering is maintained from the same argument as case (ii) discussed in Theorem 2.

For the other two cases, we first derive the range of $sn_{n_1}^{adj}$ calculated from Eq.2.1 when $msn_{(src,rreqid)}^{n_i} \geq sn_{n_1}^{n_i}$, assuming $sn_{n_1}^{rep} > sn_{n_1}^{n_i}$. Assume that $msn_{(src,rreqid)}^{n_i} = sn_{n_1}^{n_i} + \alpha$ and $sn_{n_1}^{rep} = sn_{n_1}^{n_i} + \beta$, where α, β are integers such that $\alpha \geq 0$ and $\beta > 0$. From Eq.2.1, we have $sn_{n_1}^{adj} = sn_{n_1}^{n_i} + \beta - \lfloor \frac{\beta - \alpha}{\lambda} \rfloor$, where $\lambda = wc_{(src,rreqid)}^{n_i} + 1 > 1$, and $\beta > \alpha$. Therefore , we have the inequality

$$sn_{n_1}^{n_i} \leq msn_{(src,rreqid)}^{n_i} < sn_{n_1}^{adj} \leq sn_{n_1}^{rep} \tag{2.4}$$

For cases (ii) and (iii), when $sn_{n_1}^{n_i} < sn_{n_1}^{rep}$, the route entry is updated as per Eq. 2.3, and either $sn_{n_1}^{n_i}$ is set to $sn_{n_1}^{adj}$ or incremented by one ($sn_{n_1}^{n_i} + 1$), and only increases. Hence, it is true that for any predecessor n_{i+1} that uses n_i as its successor at that instant of time that $sn_{n_1}^{n_{i+1}} < sn_{n_1}^{n_i}$, because it could not have received a higher sequence number in a RREP from n_i. With respect to node n_{i-1}: from Eq. 2.4, $sn_{n_1}^{adj} \leq sn_{n_1}^{rep} = sn_{n_1}^{n_{i-1}}$. For the RREP to be accepted, it is true that $sn_{n_1}^{rep} > sn_{n_1}^{n_i}$, and, therefore $sn_{n_1}^{n_i} + 1 \leq sn_{n_1}^{rep} = sn_{n_1}^{n_{i-1}}$. If n_i sets $sn_{n_1}^{n_i} = sn_{n_1}^{rep}$, it is also true that $d_{n_1}^{n_i} > d_{n_1}^{rep}$, because link costs are greater than zero.

Consider cases (ii) and (iii), when $sn_{n_1}^{n_i} = sn_{n_1}^{rep} \wedge d_{n_1}^{n_i} \geq d_{n_1}^{rep} + lc_{n_{i-1}}^{n_i}$. After the update, the value of $sn_{n_1}^{n_i}$ is not changed and the value of $d_{n_1}^{n_i}$ cannot increase. Hence, the ordering is not affected.

Hence, if path P exists at time t, every node n_i, where $i \in \{2, k\}$ must have accepted a RREP from n_{i-1} and updated its route entry for n_1 in one of the three states, and therefore, the theorem is true. □

Theorem 9. *At any instant, SWR is loop-free.*

Proof. From Theorem 8, the proof for loop-freedom of SWR is direct from the proof of loop-freedom of the routing-framework (Theorem 4). □

Theorem 10. *In an error-free stable connected network, a source will establish a route to a destination in finite time.*

Proof. The proof is similar to the convergence proof for the framework (Theorem 5). We give only an outline of the proof: Let source A issue a RREQ req, identified by (A, ID_A) which traverses a path, $P = \{n_k, n_{k-1}, ..., n_i\}$ (where n_i can be n_1), before reaching the destination n_1 or a node that satisfies SSC. If any node n_j requested a reset ($R_{n_1}^{req} = 1$), then the RREQ can only be answered by the destination, and every node n along the path from n_j to n_1 must have cached $reset_{(A,ID_A)}^{n} = 1$ for this computation. The RREP issued either by a intermediate node or the destination will have a sequence number greater than $msn_{n_1}^{req}$. When a node n_i updates its routing table, it is true that $msn_{(A,ID_A)}^{n_i} < sn_{n_1}^{n_i} \leq sn_{n_1}^{rep}$ (by Eq. 2.3). If node n_i is part of the path that has $reset_{(A,ID_A)}^{n_i} = 1$ cached, then it updates to $msn_{(A,ID_A)}^{n_1} < sn_{n_1}^{n_i} = sn_{n_1}^{rep}$ (by Eq. 2.2). In both cases, the new RREP is relayed with a $sn_{n_1}^{rep}$ that will satisfy ASC at node n_{i+1} because $msn_{(A,ID_A)}^{n_i} \leq sn_{n_1}^{n_{i+1}}$. If at node n_{i+1}, the RREP does not satisfy ASC , then node n_{i+1} learned a route with a higher sequence number and the new RREP generated with $sn_{n_1}^{n_{i+1}} > sn_{n_1}^{rep}$ will still satisfy ASC at the nodes along the reverse path to the source. In any case, the RREP forwarded along the reverse path will satisfy all the nodes and hence the source will be able to establish a successor path in finite time due to finite time for message exchanges. □

2.7 Performance Comparison

We present results for SWR and AODV-RSN over varying loads and mobility. The protocols used for comparison are AODV, the Dynamic Source Routing (DSR) protocol [13], and the Optimized Link State Routing (OLSR) protocol [2] which are representatives of current proposals in the MANET working group of the IETF [42]. DSR is an on-demand routing protocol that collects partial topology information in route request floods, and uses this information as source-routes that are carried in the data packets to forward the packets to the destination. OLSR is a pro-active link-state routing protocol that reduces the amount of information that must be exchanged on topology changes by having a subset of the nodes (multi-point relays) carry out the flooding of the link-state updates. Simulations are run in Qualnet 3.5.2. The parameters are set as in [41]. Both AODV and SWR use an expanding ring search during the route request flood. Additionally, we evaluate SWR and AODV with their respective local repair mechanisms, called SWR-LR and AODV-LR. We evaluate the performance loss due to forced wait periods on node reboots by comparing AODV against AODV-RSN in a demonstrative scenario where nodes reboot periodically.

2.7.1 Simulation Setup

Simulations are performed on two scenarios, (i) a 50-node network with terrain dimensions of 1500m x 300m, and (ii) a 100-node network with terrain dimensions of 2200m x 600m. Traffic loads are CBR sources with a data packet size of 512 bytes. Load is varied by using 10 flows (at 4 packets per second) and 30 flows (at 4 packets per second). The MAC layer

used is 802.11 with a transmission range of 275m and throughput 2 Mbps. The simulation is run for 900 seconds. Node velocity is set between 1 m/s and 20 m/s. We use two sets of traffic characteristics: (i) random flows have a mean length of 100 seconds, distributed exponentially, and (ii) fixed flows that last the entire simulation time. For the fixed flows, we only show results for 30-flows because we did not notice any shift in trend from the one observed for the 10-flows scenario with exponential flow distribution. Each combination (number of nodes, traffic flows, scenario, routing protocol and pause time) is repeated for nine trials using different random seeds.

2.7.2 Performance Criteria

We present four metrics. *Delivery ratio* is the ratio of the packets delivered per client/server CBR flow. *Latency* is the end to end delay measured for the data packets reaching the server from the client. *Network load* is the total number of control packets (RREQ, RREP, RERR, Hello, TC etc) divided by the received data packets. *Data hops* is the number of hops traversed by each data packet (including initiating and forwarding) divided by the total received packets in the network. This metric takes into account packets dropped due to forwarding along incorrect paths. A larger value for the data-hops metric indicates that more data packets traverse more hops without reaching the destination necessarily.

2.7.3 Performance Discussion

Tables 2.2 and 2.3 summarize the results of the different metrics by averaging over all pause times for the 50 and 100 node networks with random traffic-flows. Table 2.4 summarizes

the same set of metrics for 50 and 100-node networks with fixed traffic-flows. The columns show the mean value and 95% confidence interval. All our performance discussions focus on the average case because the confidence intervals overlap atleast slightly in most cases. The packet delivery ratio, the end-to-end delay, and the control overhead over various pause times for 50-node and 100-node networks with 30-flows is shown for random traffic-flows in Figures 2.9 and 2.7, and for fixed traffic-flows in Figures 2.10 and 2.8. The vertical bars in the graphs indicate the 95% confidence intervals.

2.7.3.1 AODV-RSN

In the 10-flow scenarios, AODV-RSN has slightly higher control-overhead, and delivery latency than AODV. This is due to the extra mechanisms required to re-learn destination sequence numbers, because of which RREQs have to be answered by the destination. However, in the 30-flow scenarios, AODV-RSN, compared to AODV, shows significantly better packet delivery, reduced control overhead and shorter latency. The performance results for AODV-RSN demonstrate that the performance of the protocol is not affected by the extra mechanisms required for maintaining correctness of the protocol.

Table 2.7.3.1 summarizes the results for the scenario that has nodes rebooting periodically every 50-second interval for the 50-nodes, 10-flows scenario. Nodes running AODV cannot participate in any routing action for $DELETE\_PERIOD$ after a reboot, whereas nodes on AODV-RSN can participate immediately. The results show the effects of forced waits on node reboots. AODV has a very low packet delivery of (0.710 ± 0.004) compared to AODV-RSN (0.995 ± 0.03) which is almost unaffected.

Nodes rebooting in AODV drop the packets if they are sources of flows or affect network connectivity. The 50-second periodic reboot is clearly unrealistic. However, the purpose of the simulations is to prove that having arbitrary wait periods after reboots can affect performance adversely.

Table 2.1: Performance average over all pause times with 10-flows in a 50-node network with nodes rebooting every 50-seconds

Metric	AODV-RSN	AODV
Delivery Ratio	0.995±0.030	0.710±0.004
Latency (sec)	0.015±0.001	0.027±0.012
Net Load	0.408±0.270	0.501±0.067
Data Hops	2.614±0.195	2.569±0.185

2.7.3.2 SWR

SWR has a very consistent performance in all scenarios and outperforms other protocols in most cases. The graphs show that SWR has excellent packet delivery in the very high mobility scenarios. OLSR and DSR show performance comparable to SWR at lower mobility. The key to the performance of SWR is the ability of many intermediate nodes to answer route requests. Hence, by using the expanding ring search for route establishment, sources can establish a route by flooding a limited diameter of the network, without requiring to flood to reach the destination as is the case of AODV. DSR performs poorly at high mobility, which can be explained due to the stale caches. OLSR has poor performance in highly-mobile or heavy-load scenarios (30-flows) where its topology information is inconsistent because of topology changes or lost and delayed control packets. Although DSR exhibits very low control overhead, it is due to the use of optimization to learn source-routes from data packets promiscously. OLSR's control overhead is independent of the traffic flows. SWR's end-to-end latency is among the lowest

across all scenarios. The data hops metric shows that, with relation to the packet delivery ratio, all protocols forward an equivalent amount of packets; however, SWR delivers more packets to the destination which reflects the correctness of the routes.

Table 2.2: Performance average over all pause times for 50 nodes network for 10-flows and 30-flows (random)

Protocol	Size	Delivery Ratio	Latency (sec)	Net Load	Data Hops
SWR	10	0.995±0.001	0.020±0.002	0.338±0.080	2.582±0.180
SWR-LR	10	0.995±0.001	0.019±0.002	0.335±0.081	2.581±0.180
AODV-RSN	10	0.989±0.004	0.024±0.006	0.336±0.081	2.589±0.189
AODV	10	0.994±0.002	0.016±0.003	0.270±0.066	2.576±0.179
AODV-LR	10	0.994±0.002	0.017±0.004	0.266±0.067	2.580±0.180
DSR	10	0.940±0.027	0.041±0.047	0.220±0.095	2.677±0.185
OLSR	10	0.887±0.040	0.012±0.001	1.937±0.220	2.456±0.175
SWR	30	0.826±0.046	0.658±0.250	2.858±0.871	2.818±0.283
SWR-LR	30	0.846±0.049	0.550±0.237	2.273±0.789	2.825±0.271
AODV-RSN	30	0.787±0.048	0.757±0.264	3.656±0.973	2.923±0.314
AODV	30	0.765±0.0553	1.010±0.356	4.423±1.289	2.951±0.324
AODV-LR	30	0.770±0.056	0.965±0.333	4.269±1.264	2.929±0.309
DSR	30	0.683±0.059	4.760±1.073	0.410±0.140	3.625±0.308
OLSR	30	0.798±0.034	0.883±0.311	0.713±0.069	2.478±0.161

Table 2.3: Performance average over all pause times for 100 nodes network for 10-flows and 30-flows (random)

Protocol	Size	Delivery Ratio	Latency (sec)	Net Load	Data Hops
SWR	10	0.989±0.004	0.053±0.010	1.423±0.402	3.757±0.317
SWR-LR	10	0.989±0.004	0.052±0.009	1.423±0.384	3.768±0.322
AODV-RSN	10	0.978±0.006	0.050±0.014	1.065±0.262	3.817±0.319
AODV	10	0.988±0.004	0.036±0.009	0.897±0.236	3.744±0.293
AODV-LR	10	0.988±0.004	0.035±0.008	0.872±0.221	3.767±0.293
DSR	10	0.876±0.050	0.099±0.057	0.859±0.353	4.257±0.317
OLSR	10	0.821±0.063	0.022±0.002	11.795±1.575	3.583±0.256
SWR	30	0.695±0.045	0.921±0.174	10.027±1.800	4.368±0.353
SWR-LR	30	0.718±0.049	0.825±0.177	8.431±1.703	4.348±0.332
AODV-RSN	30	0.660±0.038	0.897±0.171	11.776±1.980	4.545±0.363
AODV	30	0.608±0.051	1.455±0.385	18.298±13.069	4.751±0.434
AODV-LR	30	0.592±0.044	1.617±0.538	21.339±15.523	4.868±0.463
DSR	30	0.618±0.049	5.125±0.782	1.243±0.405	6.141±0.499
OLSR	30	0.821±0.063	3.371±0.532	5.423±0.669	4.014±0.277

Table 2.4: Performance average over all pause times for 50-nodes and 100-nodes network with 30-flows (fixed)

Protocol	Size	Delivery Ratio	Latency (sec)	Net Load	Data Hops
SWR-LR	50	0.813±0.069	0.570±0.328	2.852±1.131	2.905±0.350
SWR	50	0.795±0.073	0.664±0.373	3.448±1.313	2.955±0.369
AODV-RSN	50	0.729±0.075	0.826±0.393	5.588±1.689	3.122±0.417
AODV-LR	50	0.735±0.082	0.865±0.457	5.095±1.861	3.095±0.420
AODV	50	0.738±0.083	0.866±0.473	5.044±1.874	3.084±0.424
DSR	50	0.581±0.081	3.422±0.813	0.430±0.156	3.809±0.406
OLSR	50	0.772±0.042	0.842±0.413	0.727±0.072	2.534±0.187
SWR-LR	100	0.715±0.074	0.682±0.270	8.601±2.650	4.348±0.570
SWR	100	0.696±0.072	0.752±0.269	10.009±2.653	4.358±0.530
AODV-RSN	100	0.608±0.070	1.027±0.316	15.656±3.488	4.866±0.625
AODV-LR	100	0.609±0.094	1.082±0.444	17.909±12.097	4.903±0.759
AODV	100	0.608±0.088	1.060±0.402	16.812±8.743	4.731±0.696
DSR	100	0.476±0.099	3.865±1.150	1.208±0.453	6.177±0.733
OLSR	100	0.585±0.066	2.761±1.062	5.541±0.811	4.074±0.452

2.7.3.3 Local Repair

Tables 2.2, 2.3, and 2.4 list the performance metrics for AODV and SWR with local repair. AODV-LR shows very little improvement over the base performance of AODV. SWR-LR uses a simple one-hop neighbor query to repair routes locally and shows a noticeable improvement over SWR in terms of higher packet-delivery ratio, and reduced latency and control overhead. The reason for AODV's lack of improvement with the local repair optimization is due to the necessity to know the hop-count to the source, and an additional ttl-check which limits the number of repairs that can actually be initiated. The ttl-check mechanism for the local repair RREQ [33] in AODV is as follows: A local repair RREQ *est-ttl* is calculated to fall in the range, AODV_MIN_ REPAIR_TTL $<$ *est-ttl* $<$ (hopcount-to-source)/2 + AODV_LOCAL_ADD_TTL. The prescribed value for AODV_LOCAL_ADD_TTL is two, and the value for AODV_MIN_REPAIR_TTL is one. To initiate a RREQ, it is necessary that *est-ttl* be

less than the *hopcount-to-source*. Calculating the inequality, a node can initiate a RREQ only if (hopcount-to-source) > 4. This basically limits the route repair operation to only intermediate nodes that are at least at a distance of four hops from the source; if the total path length is less than four hops, no local-repair attempts are made. Intuitively, our claims correlate with a previous study on the scalability of AODV [33] where the performance results indicate that local-repair in AODV provides no additional benefits in networks of sizes 100 or less.

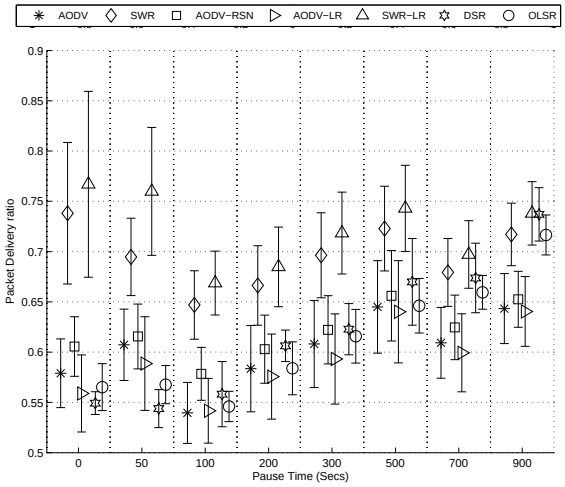
2.8 Conclusion

We have introduced new techniques for the class of routing protocols based on destination sequence numbers. Our mechanism for re-learning destination sequence numbers does not require reliable communication and ensures correct recovery after a node failure or reboot, without requiring arbitrary timers to flush out old information from the network. We also present a new sufficient condition that allows manipulation of sequence numbers by treating them as a finite label space, rather than as an absolute timestamp, effectively allowing manipulation of sequence numbers.

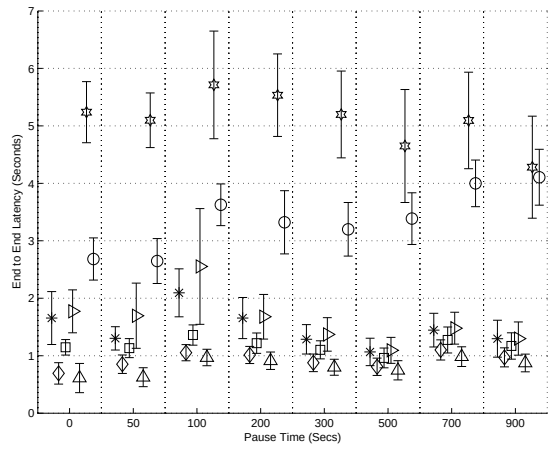
Using our safe reset technique, we have introduced a new robust and loop-free on-demand routing framework based on destination sequence numbers that can recover correctly from node failures and state loss, even when operating on top of a MAC layer such as the IEEE802.11DCF, which does not guarantee reliable message delivery or bounded message delays. Under such conditions, we have illustrated that the current AODV specification [19] is vulnerable to looping, de-facto partitions, and the counting-to-infinity problem.

We present two instantiations of our framework. We present AODV-RSN, with which we show how robustness can be added to solve AODV's problems. With SWR, we show how the framework can be used to design a robust bandwidth-efficient routing protocol that can improve performance by assigning destination sequence numbers to successor nodes along a path progressively. SWR also helps to showcase how the manipulation of sequence numbers as labels can be useful to a routing protocol. Our performance results comparing SWR and AODV-RSN against state-of-the-art MANET routing protocols (AODV, DSR, and OLSR) validate our previous claims. AODV-RSN shows that immediate node participation in the routing protocol almost leaves the performance unaffected while AODV's requirement for waiting arbitrary time periods affects performance adversely.

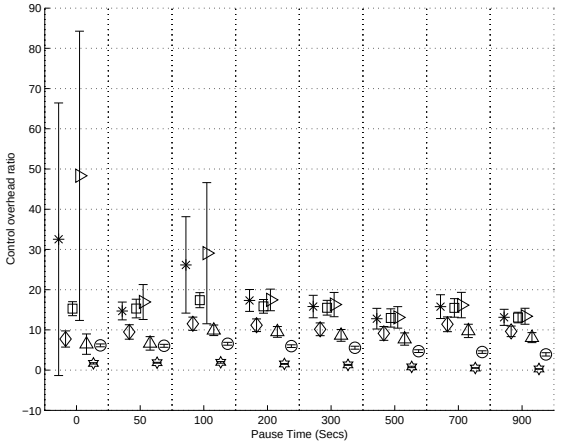
The techniques we introduce can be applied to a wider context of problems. The sequence number reset can be used to provide a robust mechanism on node failures in link-state protocols like OSPF when used in MANETs [32], pro-active destination sequence number based routing protocols such as DSDV [25], and destination sequence number based multicast routing protocols such as the multicast version of AODV [31]. The routing framework can be used to add robustness to LDR [8], the Feasible Label Routing (FLR) protocol with destination-sequenced labels [34], or the Split Label Routing (SLR) protocol [27] where the sequence number serves as a reset for distances, labels (paths), or label sets, respectively. Sequence number windows can be applied suitably to hybrid pro-active approaches such as the Adaptive Distance Vector (ADV) routing protocol [28]. The initial setup phase in ADV can be used to estimate a sequence number window increment, and by creating windows, routes can be repaired locally on-demand without waiting for the next update from the destination.



(a) Packet Delivery

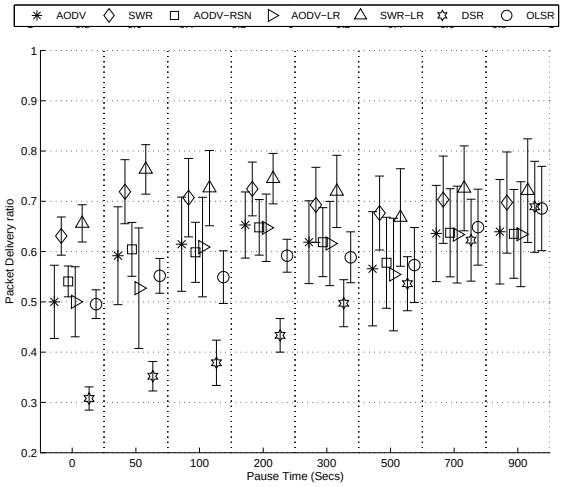


(b) Latency

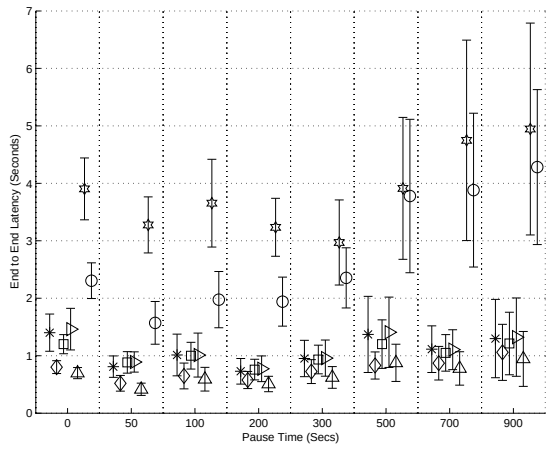


(c) Control Overhead

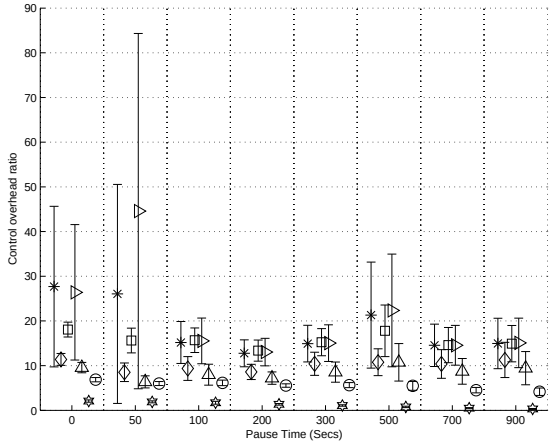
Figure 2.7: Random (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

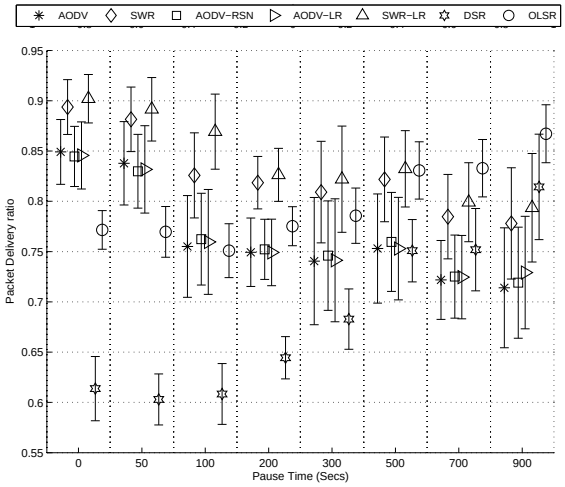


(b) Latency

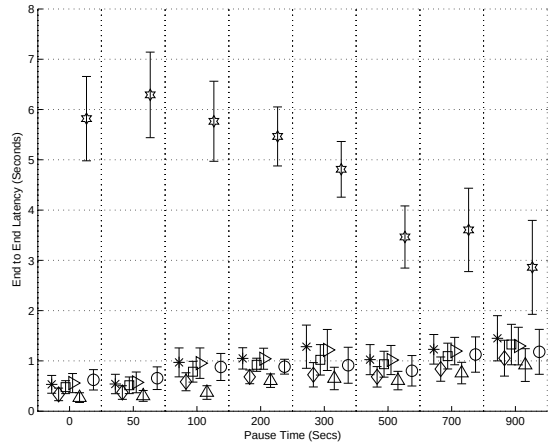


(c) Control Overhead

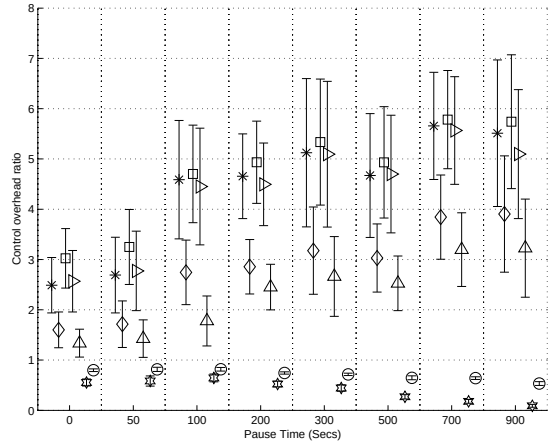
Figure 2.8: Fixed (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

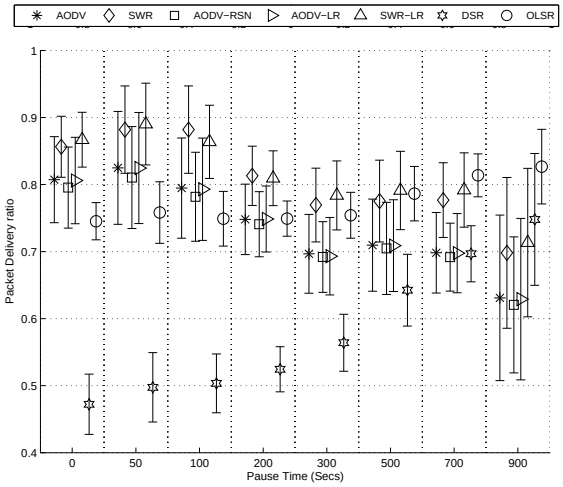


(b) Latency

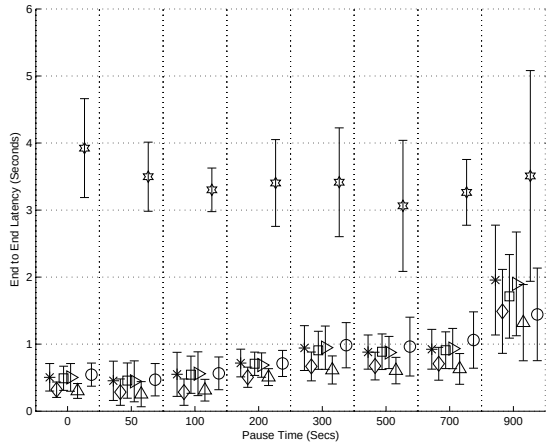


(c) Control Overhead

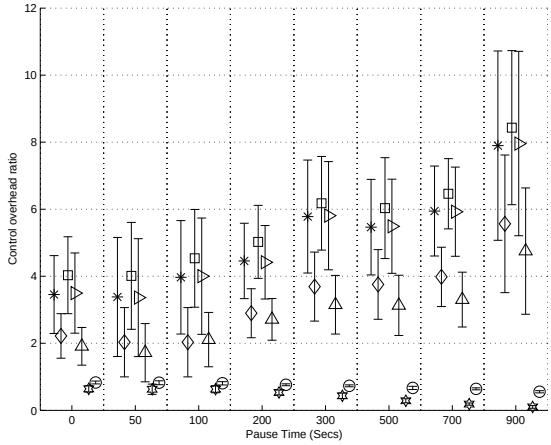
Figure 2.9: Random (50-nodes, 30-flows, 120 pps)



(a) Packet Delivery



(b) Latency



(c) Control Overhead

Figure 2.10: Fixed (50-nodes, 30-flows, 120 pps)

Chapter 3

Routing using labels as path information

3.1 Introduction

We present the Feasible Label Routing (FLR) protocol, which uses path information only in its signaling for on-demand routing, while supporting loop-free incremental forwarding of data packets when the header of each data packet simply states the address of the intended destination without requiring source-routes as in the case of prior proposals based on path information. FLR collects path information similar to other protocols, and converts the path information into labels. Just as AODV orders nodes according to increasing sequence numbers to a destination, and DUAL orders nodes according to decreasing feasible distances to a destination, FLR orders nodes according to labels that become lexicographically smaller as the destination is approached. FLR achieves this using route requests (RREQ), route replies (RREP), and route errors (RERR) that carry path information and are similar to the messaging structure of other on-demand routing protocols.

In FLR, each node maintains a label for a destination for which it needs to forward traffic, where a label is built based on the properties of a path to a destination. The *feasible label* of a node for a destination is simply the lexicographically smallest path to the destination attained by the node since the last time it had to send RERRs to its neighbors. A node can accept a RREP for a destination generated by the destination or any intermediate node if the label to the destination advertised in the reply is "smaller" than the feasible label at the node that issued the RREQ. Hence, to realize instantaneous loop freedom in FLR, RREQs carry the minimum feasible label that must be satisfied by the node issuing a RREP, and a RREP carries the *current label* of the node forwarding the reply.

Section 3.2 presents sufficient conditions for loop-free routing using feasible labels. Section 3.3 describes FLR and several optimizations. Section 3.4 summarizes how some of our results could be used to enhance AODV and DSR. Section 3.5 shows an example of FLR in operation. Section 3.6 analyzes the instantaneous loop freedom and termination properties of FLR. Section 3.7 compares the performance of FLR against AODV and DSR, which are two popular on-demand protocols, and OLSR, which is a popular a pro-active protocol. The results show that FLR attains better performance than DSR, AODV, AODV-PA, and OLSR, in terms of end-to-end delays, packet-delivery ratios, and control overhead. The better performance of FLR is due to the use of cached path information to order sources and relays with respect to destinations, rather than deriving paths to new destinations. Simulations are performed for variants of FLR that use destination-sequenced labels (labels associated with a destination-sequence number), called FLR-DL, and are similar to those used in similar to AODV-PA [11]. The results obtained for FLR-DL are far better than those of AODV-PA. This is because AODV-PA uses

path information only to setup routes to relay nodes that RREQs and RREPs traverse; however, these routes can expire quickly if there are no flows for the relay nodes and the corresponding path information is not used in future routing decisions. By contrast, the path information in FLR is stored as a label, and although it may become out-of-date, the ordering of the labels enables FLR to find a new loop-free successor to the destination on a route failure. The results for a simple neighbor-query based local repair mechanism in FLR-DL shows that the labeling is more effective in finding a neighbor as a successor compared to the local repair scheme used in AODV which requires a ttl-constrained flood to locate the destination. Section 3.8 provides our concluding remarks.

3.2 Sufficient Conditions for Loop Freedom Using Labels

We specify and prove new sufficient conditions for loop-free routing that are applicable to on-demand and pro-active routing protocols based on path information. These conditions extend the notion of ordering distances to a destination, such that nodes closer to the destination have shorter distances to it, into the notion of ordering the paths to a destination lexicographically, so that nodes closer to the destination have labels representing their paths that are lexicographically smaller.

Table 3.1 summarizes the terminology used to describe the sufficient conditions for loop-free routing, as well as to specify FLR in the next section. The values of labels, link costs, or routing-table entries are functions of time, but the time for which the values of such functions apply are specified only when needed.

A *label* L of size n, $n \geq 0$, is an ordered sequence of unique elements $\{e_1(L)\,, e_2(L),$ ...,$e_{n-1}\,(L), e_n(L)\,\}$, where each element $e_i(L)\ \in L$ consists of a valid node identifier (denoted by $e_i^{id}(L)$) and the cost of the link from $e_i(L)$ to $e_{i+1}(L)$ (denoted by $e_i^c(L)$). For the last element of L, $e_n^c(L) = 0$.

The label of node B for destination D is denoted L_D^B, and L_{DB}^A denotes the label for destination D reported by node B and stored by node A. In a routing protocol based on path information, when node A chooses node B as its successor for destination D, the path from A to D consists of the concatenation of the link from A to B with the path from B to D. Hence, if we denote the concatenation of two labels L_1 and L_2 by $L_1 \oplus L_2$, if node A chooses node B as its successor to destination D, its label equals $L_D^A = (A,\ c_B^A) \oplus L_{DB}^A$, where c_B^A is the cost of the link from A to B.

The weight of a label L of size $n\ \geq\ 0$ is denoted by the function $W(L)$, where $0 \leq W(L) \leq \infty$, and is defined by

$$W(L) = \begin{cases} \infty & \text{if } \ n = 0 \\ \sum_{i=1}^{n} e_i^c(L) & \text{otherwise} \end{cases} \tag{3.1}$$

Relational operators on two labels L_1 and L_2 of sizes n_1 and n_2, respectively, are defined as follows:

$$L_1 = L_2\ if \begin{cases} W(L_1) = W(L_2) \wedge \\ \{\ n_1 = n_2 \wedge \\ (\ e_j^{id}(L_1) = e_j^{id}(L_2)\ \forall j \mid 1 \leq j \leq n_1)\ \} \end{cases} \tag{3.2}$$

$$L_1 > L_2 \; if \begin{cases} W(L_1) > W(L_2) \vee \\ (\; W(L_1) = W(L_2) \wedge n_1 > n_2 \;) \vee \\ \{ \; (W(L_1) = W(L_2) \wedge n_1 = n_2) \wedge \\ (\; \exists j \mid \{ \; 0 < j \leq n_1 \; \wedge \; (e_j^{id}(L_1) > e_j^{id}(L_2)) \wedge \\ (\; e_h^{id}(L_1) = e_h^{id}(L_2) \; \forall \; h \mid j+1 \leq h \leq n_1) \; \} \;) \; \} \end{cases}$$

$$L_1 < L_2 \; if \begin{cases} W(L_1) < W(L_2) \vee \\ \{ \; W(L_1) = W(L_2) \wedge n_1 < n_2 \; \} \vee \\ \{ \; (\; W(L_1) = W(L_2) \;) \; \wedge \; (n_1 = n_2) \wedge \\ (\exists j \mid \{ \; 0 < j \leq n_1 \; \wedge (\; e_j^{id}(L_1) < e_j^{id}(L_2) \;) \wedge \\ (\; e_h^{id}(L_1) = e_h^{id}(L_2) \; \forall \; h \mid j+1 \leq h \leq n_1 \;) \; \} \;) \; \} \end{cases}$$

The following sufficient conditions for loop-free routing using labels extend the conditions introduced for DUAL, which were based on distances to destinations [7]. The new conditions assume that (a) nodes communicate to neighbors their labels to destinations proactively or on demand, and (b) each node maintains a routing table specifying the next hop(s) to some or all destinations in the network, and the labels notified by other nodes for some or all destinations.

The label reported by node B for destination D and stored at node A is denoted by L_{DB}^A, and L_D^{*A} and L_{DB}^{*A} denote the minimum values attained by L_D^A and L_{DB}^A, respectively.

Feasible Label Condition (FLC): Node A can make node B its successor for destination D after processing an input event if $L^A_{DB} < L^{*A}_D$. If no neighbor exists with a smaller reported label than L^{*A}_D, then node A must keep its current successor if it has any.

Extended Label Condition (ELC): Node A can make node B its successor for destination D after processing an input event if $(A,\ c^A_B) \oplus L^A_{DB} < L^{*A}_D$, where $(A,\ c^A_B)$ states the identifier of A and the cost of the link from A to B. If no neighbor exists with a smaller reported label than L^{*A}_D, then node A must keep its current successor if it has any.

Next-hop Label Condition (NLC): Node A can make node B its successor for destination D after processing an input event if $L^A_{DB} < L^{*A}_{DS}$, where S is node A's current successor to destination D. If no neighbor exists with a smaller reported label than L^{*A}_{DS}, then node A must keep its current successor if it has any.

Theorem 11. *Using FLC whenever nodes choose their successors to destination D is sufficient to ensure that no routing-table loops are created for destination D.*

Proof. FLC is equivalent to SNC from DUAL, which is shown to be loop-free [see [7], Theorem 1, pp. 132ff]. FLC uses feasible labels, which simply extrapolate the ordering properties of feasible distances used in DUAL by using the relational operators on labels stated in Eq. 3.2. □

As Theorem 11 shows, using FLC guarantees that routing-table loops are not created, and the other conditions can also be shown to be sufficient to ensure loop-free routing using

Table 3.1: Terminology used for FLR

Notation	Description
L_D^A	The stored label for destination D at node A.
s_D^A	The successor for destination D at node A.
L^∞	An invalid route or a label of infinite cost.
FL_D^A	The smallest label assigned by node A for D since A sent its last route error for D.
PS_D^A	The set of neighbors of node A to whom node A has sent RREPs for D.
c_B^A	The cost of the link from node A to neighbor B.
rep	Superscript used for variables in a route reply.
req	Superscript used for variables in a route request.

labels. However, nodes must keep choosing as successors to destinations those neighbors that offer labels that are always "smaller" than the smallest labels they have attained. Consequently, even if there are physical paths from node A to destination D, it is possible for node A to be unable to pick any neighbor as successor to D if FL_D^A is not larger than any label reported by a neighbor of node A.

In practice, additional mechanisms are needed together with one of the sufficient conditions for loop freedom stated above to allow nodes to increase their feasible labels to destinations safely (without causing loops). One approach to allowing a node to safely increase its feasible label for a destination would be the use of diffusing computations as in DUAL. However, a diffusing computation is impractical in MANETs, because it requires the origin of the computation to coordinate the updating of its successor for a given destination with nodes many hops away whose paths to the destination include the origin of the computation. The next section introduces FLR, which implements much more efficient mechanisms for allowing nodes to increase their feasible labels to destinations, without creating routing-table loops.

3.3 Feasible Label Routing Protocol (FLR)

3.3.1 Principles of Operation

FLR uses route request (RREQ), route reply (RREP), and route error (RERR) messages similar to that of other on-demand routing protocols. The routing-table entry at node A for destination D includes the current label (L_D^A), the feasible label (FL_D^A), the successor (s_D^A), and the predecessor set for D (PS_D^A). Labels are as defined in Section 3.2, and FL_D^A is the lexicographically smallest label that node A has obtained for D since it sent its last RERR for D. PS_D^A is the set of predecessors for destination D, which are those neighbors of node A to whom node A has sent RREPs for D. The superscripts req and rep are used for variables included in a RREQ and RREP, respectively.

FLR is based on the following four rules (called conditions), which apply for a given destination D independently of other destinations. These rules are used to implement FLC in an on-demand routing context and to permit nodes to increase their feasible labels when needed, without causing routing-table loops. The rules make use of the minimum feasible label of any of the nodes that relayed or originated a RREQ for destination D (denoted by MFL_D^{req}), the path traversed by the RREQ (denoted $path_D^{req}$), and the current label of the node that transmits a RREP for D (denoted by L_D^{rep}).

*A*LC: (Accept Label Condition). When node A receives a RREP from node B for destination D with $L_D^{rep} \notin A$, then

If No Local Repair Used:

Node A sets $s_D^A \leftarrow B$ if $L_D^A = L^\infty$ or ($L_D^A \neq L^\infty$ and $(A,\ c_B^A) \oplus L_D^{rep} < L_D^A$ and

$L_D^{rep} < FL_D^A$).

If Local Repair Used:

Node A sets $s_D^A \leftarrow B$ if $((A,\ c_B^A) \oplus L_D^{rep} < L_D^A$ and $L_D^{rep} < FL_D^A)$. Node A sends a RERR reliably to its neighbors in PS_D^A and then sets $s_D^A \leftarrow B$ and $PS_D^A \leftarrow \phi$ if $((A,\ c_B^A) \oplus L_D^{rep} < L_D^A$ and $L_D^{rep} \not< FL_D^A)$.

*S*LC: (Start Label Condition). Node I can issue a RREP responding to a RREQ for destination D if I has an active route to D and $L_D^I < MFL_D^{req}$.

*M*LC: (Minimum Label Condition). If node A relays a RREP for destination D, it sets $L_D^{rep} = L_D^A$. Node A relays a RREQ for destination D only if $A \notin path_D^{req}$ and sets $MFL_D^{req} = min\{MFL_D^{req}, FL_D^A\}$.

*R*LC: (Reset Label Condition). If node A must change s_D^A, then it sets $L_D^A \leftarrow L^\infty$ and

If No Local Repair Used: Node A sends a RERR reliably to its neighbors in PS_D^A before setting $PS_D^A \leftarrow \phi$ and sending a RREQ for D with $MFL_D^{req} = FL_D^A$.

If Local Repair Used:

Node A sends a RREQ containing $MFL_D^{req} = FL_D^A$.

When no local repairs are used, node A first sends a RERR to block those neighbors using A as next hop to D with a reliable RERR, and then it attempts to obtain a new next hop to D with a RREQ (RLC). A RREQ traverses loop free paths and carries the minimum feasible label of any of its relays (MLC), and only a node with a label strictly smaller than the minimum feasible label of the RREQ can create a RREP (SLC), and a RREP which specifies the label

of the node forwarding it (MLC). If local repairs are used, node A attempts to obtain a new successor with a label smaller than its own feasible label, and blocks those neighbors using A as next hop to D with a reliable RERR if no such neighbor is found and its feasible label has to be changed.

3.3.2 Information Stored and Exchanged

The routing information used by nodes running FLR is maintained in the routing table. The routing-table entry for a destination D at a given node A specifies: (a) The successor to D (s_D^A), (b) the predecessor set for D (PS_D^A), (c) the current path label (L_D^A), and (d) the feasible label for D (FL_D^A).

A RREQ consists of the tuple $\{dst, src, reqid, MFL_{dst}^{req}, path_{dst}^{req}\}$, where src is the identifier of the source of the RREQ seeking a path to the destination dst. The $reqid$ is an identifier assigned by src to the RREQ, such that the pair $(src,\ reqid)$ is unique. The $path_{dst}^{req}$ field is a list of $\{id,\ c\}$ pairs specifying the nodes (id) that were traversed by the RREQ and the associated link cost c of each hop.

A RREP consists of the tuple $\{dst, src, L_{dst}^{rep}, ttl, path_{dst}^{rep}\}$. The field src specifies the origin of the RREQ that caused the RREP. The field ttl is the time remaining for the route to dst at the node transmitting the RREP. The label L_{dst}^{rep} is the current label of the node transmitting the RREP. The field $path_D^{rep}$ is a list of $\{id,\ c\}$ pairs specifying the nodes (id) that were traversed by the RREQ that originated the RREP and the associated link cost c of each link.

A RERR message consists of the tuple $\{orig, reset\}$. The field $orig$ is the node generating the route error message. The field $reset$ is the list of destinations for which the origin

of the RERR must reset its feasible label, and informs the recipients of the RERR that its origin needs a new route for each destination listed in $reset$.

3.3.3 Basic Route Maintenance

3.3.3.1 Initiating a RREQ

Node A is said to be *active* for the route computation for destination D (i.e., the RREQ) that is uniquely identified by the pair (A, ID_A) when it originates such a RREQ. A node can be the origin of at most one RREQ for the same destination at any given time. The RREQ (A, ID_A) terminates when either node A attains a feasible label for destination D (by receiving a RREP for its RREQ) or the timer for its RREQ expires.

A node A that requires a route for destination D buffers the data packets if it is active for destination D. Otherwise, it issues a RREQ $\{D, A, reqid = ID_A, MFL_{dst}^{req}, path_D^{req}\}$ by setting $ID_A \leftarrow$ *incremented request counter*; $reqid \leftarrow ID_A$; $MFL_D^{req} \leftarrow FL_D^A$; $path \leftarrow \phi$; and *RREQ timer* $\leftarrow$ *(*2.ttl.latency), where ttl is the time-to-live of the broadcast flood and latency is the estimated per-hop latency of the network.

If node A receives no RREP for destination D after the expiry of its timer for RREQ (A, ID_A), it sends a new RREQ with an increased ttl. If after a number of attempts node A does not receive a RREP, a failure is reported to the upper layer. The number of hops that a RREQ can traverse is controlled externally from the RREQ by means of the TTL field of the IP packet in which a RREQ is encapsulated, or by other means.

3.3.3.2 Relaying RREQs

A node relaying a RREQ originated by another node is said to be *engaged* in the RREQ. A node may be engaged in multiple RREQs for the same destination, and a node that engaged in a RREQ maintains no explicit state for it.

When node B receives RREQ $\{D,\ A,\ \ reqid = ID_A,\ \ MFL^{req}_{dst}, path^{req}_D\}$, it first determines its own status for (A, ID_A). If B is active (i.e., $B = A$) or engaged (i.e., B is listed in $path^{req}_D$) in the computation (A, ID_A), it silently drops the RREQ. Otherwise, node B is said to be passive. If this is the case and SLC is satisfied (i.e., $L^B_D < MFL^{req}_D$), then node B issues a RREP (Section 3.3.3.4). Else, if SLC is not satisfied, node B relays the RREQ with $MFL^{req}_D \leftarrow min\{FL^B_D,\ MFL^{req}_D\}$ and $path^{req}_D \leftarrow (B,\ c^B_D) \oplus path^{req}_D$.

3.3.3.3 Route Failures

Node A sets $PS^A_D \leftarrow \phi$, $s^A_D \leftarrow nil$, $L^A_D \leftarrow L^\infty$ if no data packets have been forwarded using this route entry for $active\_route\_time$ seconds.

Node A carries out the following steps if its predecessor set (PS^A_D) is not empty and either node A receives a link-level notification that its link to s^A_D has failed, or node A receives a RERR from s^A_D:

No Local Repair Used: Node A sends a RERR reliably to all the nodes in PS^A_D, sets $PS^A_D \leftarrow \phi$, and after taking those steps it sends a RREQ for D (Section 3.3.3.1) if node A is a source of data packets for D.

Local Repair Used: Node A originates a RREQ for D (Section 3.3.3.1) and waits

for a RREP. If node A receives a RREP for D, then it proceeds as stated in Section 3.3.3.5.

3.3.3.4 Initiating and Processing RREPs

When node I processes RREQ $\{D, A, reqid = ID_A, MFL_D^{req}, path_D^{req}\}$ and SLC is satisfied (i.e., $L_D^I < MFL_D^{req}$), it issues a RREP $\{D, src, L_D^{rep}, ttl, path_D^{rep}\}$ where $L_D^{rep} \leftarrow L_D^I$. The destination or the intermediate node initiating the reply sets $path_D^{rep}$ to the reverse path $path_D^{req}$ traversed by the RREQ.

If node A receives RREP $\{D, src = S, L_D^{rep}, ttl, path_D^{rep}\}$, it determines if it is the source S of the RREQ that caused the RREP. If so, it proceeds as Section 3.3.3.5 describes. If $A \neq S$, then it updates its routing table as Section 3.3.3.5 states, and forwards the RREP along $path_D^{rep}$ setting $L_D^{rep} \leftarrow L_D^A$. Node A also adds the next hop of the RREP, B, to PS_D^A.

3.3.3.5 Adding and Updating Routes

By definition, if node A has no routing-table entry for destination D, its label and feasible label for D are assumed to have infinite cost. If link (A, B) changes its cost c_B^A, then for each destination D for which $s_D^A = B$, node A updates c_B^A in L_D^A and $FL_D^A \leftarrow min\{FL_D^A, L_D^A\}$.

When Node A receives RREP $\{D, S, L_D^{rep}, ttl, path_D^{rep}\}$ from neighbor B, then it sets a temporary label $TL_D^A \leftarrow (A, c_B^A) \oplus L_D^{rep}$ and carries out the following steps:

No Local Repair Used: If $L_D^A = L^\infty$ or $L_D^A \neq L^\infty \wedge TL_D^A < L_D^A \wedge L_{rep}^A < FL_D^A$, then node A sets $s_D^A \leftarrow B$, $L_D^A \leftarrow TL_D^A$, and $FL_D^A \leftarrow min\{FL_D^A, TL_D^A\}$.

Local Repair Used:

1. If $L_D^{rep} < FL_D^A$ and $L_D^A > TL_D^A$, then node A sets $L_D^A \leftarrow TL_D^A$, $FL_D^A \leftarrow min\{FL_D^A, TL_D^A\}$, and $s_D^A \leftarrow B$.

2. If $L_D^{rep} \not< FL_D^A$ and $L_D^A = L^\infty$, then node A sends a RERR reliably to all the nodes in PS_D^A, and then sets $PS_D^A \leftarrow \phi$, $s_D^A \leftarrow B$, $L_D^A \leftarrow TL_D^A$, and $FL_D^A \leftarrow min\{FL_D^A, TL_D^A\}$.

3.3.4 Route Maintenance Optimizations

Several optimizations can be added to FLR's basic operation, in terms of how cached labels and path information included in RREQs are used.

3.3.4.1 Inferring Paths to Relays

The labels already stored in the routing tables for some destinations can be used to infer paths to other destinations that appear as relays in such labels. Because FLR assumes that data packets are forwarded incrementally (hop by hop) using only the addresses of the intended destinations, a node can forward a data packet for a destination to a neighbor only if that neighbor has a valid route to that destination, for otherwise the packet would simply elicit a RERR.

FLR can be extended with Source Routed Requests (SRREQ) that carry the path along which the request is forwarded before it is answered by a relay or the destination. Sending a SRREQ avoids flooding the network with RREQs. The path to a destination is determined by executing a path selection algorithm (i.e., Dijkstra) over the current set of all active labels (paths) in the routing table.

3.3.4.2 Multi-path Routing

The use of labels is conducive to allowing multi-path routing. Two approaches can be followed for multi-path routing using FLR, and both require a node to store the label reported by each of its neighbors for a given destination. In FLR, node A should store the label for destination D reported by its neighbor B (denoted by L^A_{DB}) only if $L^A_{DB} < FL^A_D$.

In one approach, a node with multiple paths to a destination uses all of its available paths balancing the traffic load among them. The advantage of this approach is that it can be used to reduce packet-delivery delays (e.g., [23]). In another approach, a node uses only one path for a destination and uses other paths when the currently used path is lost. Because paths not being used may no longer be active in one or more of the relays along the path, a node that needs to use an alternate path must send a SRREQ in much the same way as when a node infers paths from cached information. The SRREQ is answered by the first relay with an active route to the destination.

3.3.4.3 Using Reverse Paths

A RREQ initiated by a node can be used by nodes receiving the request to update their routing tables with a route to the source of the RREQ in the reverse direction. However, because of the necessity for accurate predecessor information, the route cannot be used for data packet forwarding. A route entry created from the reverse route in a RREQ is valid for the reverse-route ttl, and a SRREQ is used to validate the path before data packets are forwarded. This allows the predecessor set to be built at nodes along the path when the RREPs to the SRREQs are received.

3.3.4.4 Keeping Labels Unique

We note that the basic operation of FLR does not allow a node A to accept a RREP from a neighbor B in which $A \in L_D^{rep}$. As an example of this case, consider node A with $L_D^A \ni \{X \oplus \ldots \oplus D\}$ and assume that node X changes its label and feasible label to D to $FL_D^X = L_D^X > L_D^A$ after blocking its predecessors for D. If node X needs to find a new path to D and sends a RREQ, node A may be able to reply to X's RREQ if L_D^A is smaller than the minimum feasible label of the RREQ created by X. The reply from A would provide X with a label $L_D^x \ni \{X \oplus \ldots \oplus D\}$. Obviously, node A should not provide X with a label that already includes X, because that indicates to A that it has outdated path information and the RREP will be dropped by X.

Hence, in general, if node A has an active route to destination D and label L_D^A, it should not reply to any RREQ if $\exists j |\ 0 < j \leq |path^{req}| \wedge \ \ (e_j^{id}\ (path^{req}) \in L_D^A \ \vee \ \ src^{req} \in L_D^A)$, where $|path^{req}|$ is the number of elements in the label.

3.3.4.5 Using Destination-Sequenced Labels

We define a *destination-sequenced label* DL as the union of a label L and a sequence number assigned to L by the last element of the label, denoted by $SN(L)$. The relational operators for two labels defined in Eq 3.2 can be extended to destination-sequenced labels. Let $DL_1 = L_1 \cup SN(L_1)$ and $DL_2 = L_2 \cup SN(L_2)$, then

$$DL_1 = DL_2 \; if \; \{\; SN(L_1) = SN(L_2) \; \wedge \; L_1 = L_2 \;\} \tag{3.3}$$

$$DL_1 > DL_2 \, if \begin{cases} SN(L_1) < SN(L_2) \vee \\ \{ SN(L_1) = SN(L_2) \wedge L_1 > L_2 \} \end{cases}$$

$$DL_1 < DL_2 \, if \begin{cases} SN(L_1) > SN(L_2) \vee \\ \{ SN(L_1) = SN(L_2) \wedge L_1 < L_2 \} \end{cases}$$

With the relational operators of Eq. 3.3, the same sufficient conditions for loop-free routing using labels introduced in Section 3.2 can be shown to also hold for destination-sequenced labels. Hence, FLR can be based on destination-sequenced labels and destination-sequenced feasible labels.

To take advantage of sequence numbers as part of labels, SLC must be modified to allow the destination to increase its own sequence number each time it answers a RREQ:

SLC-DL: (Start Label Condition for Destination-Sequenced Labels). Node I can issue a RREP responding to a RREQ for destination D if I has an active route to D, and $DL_D^I < MFDL_D^{req}$. If $I = D$, then node I increments $SN(L_D^D)$ before it sends its RREP (which specifies $DL_D^{rep} = L_D^D$) over the reverse path traversed by the RREQ.

There are two different approaches that can be followed when handling RREQ's.

Approach 1: RLC *can* be modified into an RLC with destination sequenced labels (RLC-DL), so that when node A has to originate a RREQ it increments the sequence number of the minimum destination sequenced feasible label ($MFDL$) sent in the RREQ. Doing so amounts to a generalization AODV's sequence numbering approach. The advantage of modifying RLC this way is that, as we state in the next section, no packet filtering based on prede-

cessors is needed to prevent data packets from looping when RERRs are sent unreliably. The disadvantage is that fewer intermediate nodes are able to satisfy SLC-DL and generate a RREP.

Approach 2: More intermediate nodes can satisfy SLC-DL if nodes originate a RREQ following RLC with the stored destination sequenced label for the $MFDL$. Because the destination sequence number associated with the label is not incremented, this scheme allows more nodes to satisfy SLC-DL. Intermediate nodes storing a DL with the same sequence number as the $MFDL$ of the RREQ can answer if they have a stored lower cost label for the destination, or if they have a higher sequence number. Intuitively, the sequence numbers associated with the labels serve as a pseudo time-stamp to determine the freshness. Within the same time-stamp the labels are ordered lexicographically, and the time-stamps of the labels get fresher along the path towards the destination. To maintain the strict ordering, the following additional modifications are required to the signalling scheme: RREQs and RREPs to carry a reset ('R') bit and a Destination-Initiated ('D') bit, respectively. The MLC needs to be modified into a MLC with destination-sequenced labels (MLC-DL) where if $MFDL$ in the RREQ is fresher (i.e., higher sequence number or same sequence number and higher cost label) than the stored FL for the destination, then the 'R'-bit must be set when relaying the RREQ. Intermediate nodes in SLC-DL must generate RREPs only if 'R' bit is not set. The 'D'-bit must be set when the destination originates the RREP, and nodes with no known $MFDL$ must only accept RREPs with 'D'-bit set. This is for the purposes of correct termination when per-destination sequence numbers are used and is discussed in [14], [15]. This approach is similar to LDR, which was proposed as an improvement on AODV and allows more intermediate nodes to reply. In LDR, sequence numbers serve as a "timestamp" for the distances to determine the freshness.

The destination-sequenced label scheme based on *Approach 2* for re-establishing routes lends itself to a simple *local repair* scheme without any constraints (i.e., information about hop-count to source, ttl-checks, etc.,) as in the case of AODV. An intermediate node experiencing a link failure for a destination will send a RREQ as per RLC with ttl set to 1. The node buffers the data packets until it receives a reply. If the RREQ timer expires, the node sends a RERR as per default rules.

Approach 3: Given that destination sequence numbers are maintained for each destination and that including the sequence numbers of the relays along the path to a destination is straightforward, the notion of a destination-sequenced label can be extended to take into account the sequence number assigned to each node along the path. In this case, the relational operators defined above can be modified to include a comparison of the sequence number of each node in the labels being compared. For example, consider the following two extended destination-sequenced labels with the sequence numbers of relay nodes, EDL_1=$\{[A,1], [B,1], [C,1], [D,1]\}$, and EDL_2=$\{[F,1], [E,1], [C,2], [D,1]\}$. Note that the new comparison based on sequence numbers will result in EDL_2 being considered "fresher" than EDL_1 (i.e., $EDL_2 < EDL_1$), whereas, it would have been considered "older" without the extended sequence numbers. The extended labels implicitly capture the freshness of the paths themselves, because a sequence number associated with every relay node can double as a link sequence number. This will prove useful for previously discussed optimizations based on topology information.

A limitation with any optimization of FLR based on destination sequenced labels is that sequence numbers must be handled carefully when nodes reboot, a network partitions, and sequence numbers wrap around. However, we have proposed approaches that solve these

robustness issues [14] [15].

3.3.5 Forwarding Data Packets

Thus far we have stated that RERRs are sent reliably to the predecessors of a node. However, achieving reliable transmissions of control packets intended for multiple destinations is not practical in MANETs that rely on contention-based MAC protocols (e.g., IEEE 802.11 DCF). Hence, the operation of FLR needs to be modified slightly to accommodate an unreliable link layer.

A simple way to accommodate an unreliable MAC protocol in FLR consists of allowing nodes to broadcast RERRs unreliably, and requiring nodes to determine if a packet that needs to be forwarded was received from a predecessor or not. Because a node that sends a RERR for a destination empties its predecessor set for the destination, it follows that no packets can traverse loops if nodes forward packets only when they are received from valid predecessors. Accordingly, when node A receives a data packet for destination D from neighbor B, node A forwards the packet to its own next hop for destination D if the node has an active route for D and $B \in PS_D^A$. Otherwise, node A drops the packet and sends a RERR for destination D to node B.

Packet filtering is not needed if destination-sequenced labels are used and nodes that originate RREQs increase the sequence number for the destination in the MFDL carried in the RREQ. This is because nodes that use a given node A in their paths to a destination are unable to satisfy SLC-DL unless they have a destination-sequenced label with a sequence number that is larger than the one in the RREQ.

3.4 Application to DSR and AODV

FLR can be simplified by using node labels directly instead of feasible labels. With such simplification, a node simply maintains its current label, next hop and predecessor set for a destination; a RREQ carries the smallest label of any of the nodes that relayed the RREQ (ML_D^{req}) instead of MFL_D^{req}; and ALC, SLC, MLC and RLC are implemented using ML_D^{req} instead of MFL_D^{req}. This simplification is practical when links have unit cost only.

The above simplification can be used in two ways related to DSR. One approach consists of using the simplified signaling of FLR, which essentially adds ML_D^{req} in RREQs to DSR's signaling and requires using ALC, SLC, MLC and RLC, together with source-routed data packets. The other approach consists of using FLR's simplified signaling together with the packet filtering described in Section 3.3.5, so that loop-free packet forwarding can be enforced using only the destinations of the data packets on a hop-by-hop basis.

The simplified FLR can also be used in the context of a "path-oriented" AODV embodiment in which destination sequence numbers are not used, and path information is used together with ALC, SLC, MLC, RLC. This application of FLR requires that nodes implement data-packet filtering based on predecessor information.

The performance results presented in Section 3.7 for networks in which all link costs are equal are indicative of how a simplified FLR approach would perform. The main reason why FLR performs so much better than DSR and AODV is that nodes apply SLC to reply to RREQs and ALC to accept RREPs in FLR and not in DSR and AODV. In DSR, this results in source routes leading to relays that must drop packets, and in AODV this translates into forcing

the destinations to reply to RREQs more often than in FLR.

If data-packet filtering based on predecessor information cannot be implemented in a MANET in which RERRs are delivered unreliably, then another variant of a simplified FLR can be adopted to obtain a "path-oriented" AODV. Specifically, FLR based on the above simplification, destination-sequenced labels, and RLC-DL can be used in the context of AODV with path accumulation (AODV-PA) [11].

The performance results reported for AODV-PA by Gwalani et al. [11] indicate that AODV-PA and AODV have similar packet-delivery ratios, while AODV-PA provides slightly better performance than AODV in terms of normalized routing load and delays. On the other hand, our results in Section 3.7 show that FLR with destination-sequenced labels provides substantial performance improvements over AODV-PA, and over the base version of FLR. Hence, it is apparent that the use of destination-sequenced labels not only allows intermediate nodes to be able to reply to RREQs but also is able to determine the freshness of the labels.

3.5 FLR Example

Figure 3.1(a) shows the directed acyclic successor graph for destination E in a six-node network. The feasible labels for destination E at each node are shown for time t_1 next to each node. Nodes A, B, D and F have active flows for node E. The label for destination E at the nodes is not shown separately, because the feasible labels are equal to the assigned labels.

Node C does not have an active route to node E and therefore has an invalid label L^{∞}. The labels marked in the figure are composed of tuples of *(id, cost)*. The predecessor sets

maintained at nodes F and B are $PS_E^F = \{B\}$ and $PS_E^B = \{A\}$, respectively.

3.5.1 Update Activity

Link e_2 fails at time $t > t_1$, and node B broadcasts a RERR. This RERR may not be received by A due to the unreliability of the MAC layer, but node B removes A from PS_E^B. Node B now initiates a new computation (B, ID_B) searching for a route for E using a feasible label $\{(B,1), (F,1), (E,0)\}$.

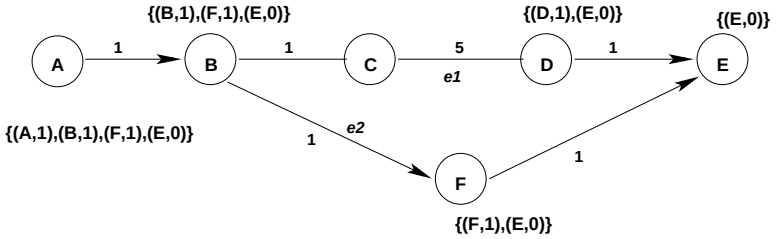
When node A receives B's RREQ, it cannot satisfy SLC and must relay the RREQ. The RREQ relayed by A is dropped by B, because it cannot engage itself again in computation (B, ID_B). B's RREQ is relayed by node C, because node C does not have an active route and therefore SLC cannot be satisfied.

Node D generates a RREP when it receives the RREQ relayed by C, because it satisfies SLC. The RREP from D carries the label $L_E^{rp_D} = \{(D,1), (E,0)\}$. Node C can accept this RREP, and updates its routing table, setting $L_E^C = FL_E^C = \{(C,5), (D,1), (E,0)\}$. Node C adds node B to its predecessor set PS_D^C for D and forwards the RREP carrying the label $\{(C,5), (D,1), (E,0)\}$.

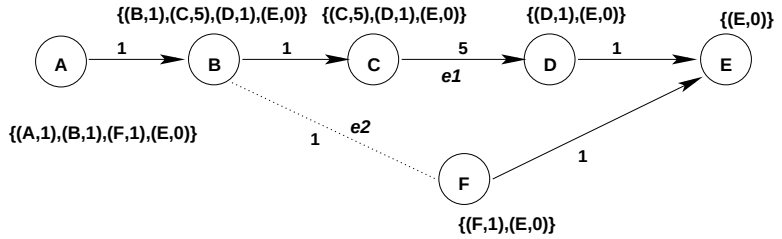
When node B receives the RREP from C, it updates its routing table and makes $s_E^B = C$. Given that B already sent a RERR for destination E to is predecessors, it can reset its feasible label. Accordingly, node B sets L_E^B and FL_E^B to $\{(B,1), (C,5), (D,1), (E,0)\}$. The feasible labels for destination E at time t_2 are shown in Fig.3.1(b).

If on the other hand, B was performing a local repair, then it would have sent a RERR to A only after processing the RREP from C. The RERR would not be sent if C had offered a

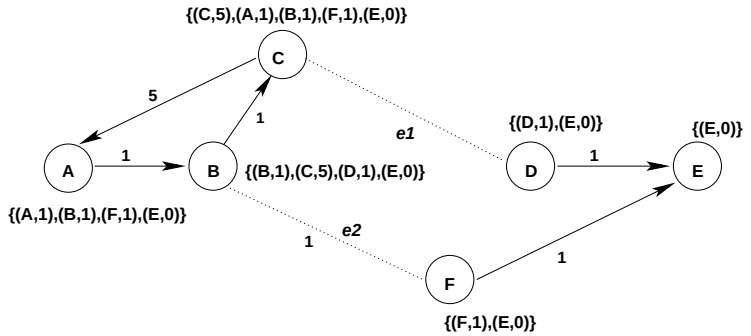
label satisfying ALC with local repair, in which case A would not have received a RERR from B.



(a) Time t_1

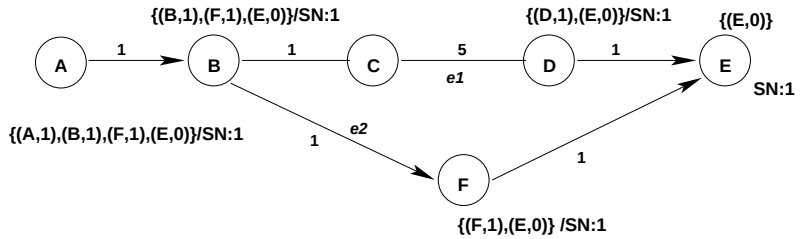


(b) Time t_2

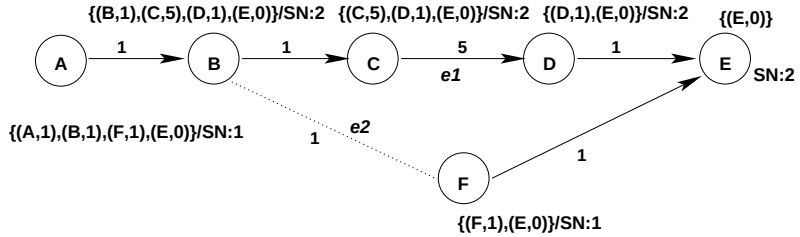


(c) Time t_3

Figure 3.1: Illustration of FLR.



(c) Time t_1



(c) Time t_2

Figure 3.2: Illustration of FLR with destination-sequenced labels

3.5.2 Loop Detection

To illustrate loop detection in FLR route errors are sent unreliably, let us assume that the RERR sent by B was not received by node A. Node A still considers node B as its successor for E according to its routing table, and the ordering of the feasible labels along the path to E has been violated due to the undelivered route error.

Assume that node C becomes a source for destination E time $t_2 + \epsilon$. Because node C has an active route to E, a new RREQ is not started, and if forwards data packets to its next hop D. Now, at time $t > t_2 + \epsilon$, link e_1 fails. Node C detects the link failure and notifies B through a broadcast RERR that is not received by B. Node B maintains C as its next hop to destination E, and node C has removed node B from PS_E^C.

Now node C starts a new route computation (C, ID_C) and issues a new RREQ with

feasible label $\{(C,5),\ (D,1),\ (E,0)\}$. Node A satisfies SLC with a lower cost label $\{(A,1),\ (B,1),\ (F,1),\ (E,0)\}$ and issues a RREP with its current label for E. Node C accepts A's RREP because it satisfies ALC and makes A its successor for destination E, creating a routing-table loop among nodes A,B and C.

Fig. 3.1(c) shows the directed successor graph for destination E at time t_3 after node C updates its routing table. However, this loop cannot persist, because A is no longer in the predecessor list of B, which forces B to drop any data packet it receives from A and to send a RERR to A in each case. If A never sends a data packet, then the routing-table loop is broken when node A's route entry expires. The same argument applies between nodes B and C.

3.5.2.1 Destination-Sequenced Labels

Fig. 3.2(d) shows the state of the network at time t_1 following the same sequence of events as in the previous example of loop-detection when FLR is optimized with destination-sequenced labels as described in Section 3.3.4.5 using *Approach 2*. Here, nodes store a destination-sequence number for D denoted by D in-addition to their label . When link e_2 fails, node B initiates a new RREQ computation with an $MFDL\ =\ [SN = 1,\ \{(B,1),\ (F,1),\ (E,0)\}]$. However, node C cannot answer it and because it violates the ordering (i.e., $L_E^C = \infty$). The reset bit 'R' in the RREQ is set when node C relays the RREQ and forces node D to relay as well. Node E increments its SN from 1 to 2 on receiving the RREQ because the 'R'-bit is set and the MDFL carries a $SN = 1$, and issues a RREP with $DL\ =\ [SN = 2, \{D\}]$ and the 'D'-bit set. Node D can accept the RREP because $SN_E^{rep} = 2$. Similarly, node D accepts the RREP as the 'D'-bit is set and forwards it to B. Fig. 3.2(e) shows the labeling of the nodes at

time t_2 when B re-establishes a route to E.

To show how FLR with destination-sequenced labels can avoid the formation of temporary loops even when route errors cannot be unreliably delivered, assume that node B's RERR has not been delivered to A at time t_2 and node A still considers B as next hop to E. Note that even though the ordering of the FL's is violated, the associated destination sequence numbers increase along the path towards the destination and the ordering of the labels is still valid when the sequence numbers are equal. As in the previous example of loop-detection, if link $e1$ fails, node C cannot accept a DL from node A because $SN_E^C = 2 > SN_E^A = 1$. The destination-sequence number associated with the label allows nodes to determine which labels to trust and hence no loops are formed.

3.6 Analysis of FLR

We prove that FLR as described in Sections 3.3.1 to 3.3.3 is loop-free at every instant if reliable RERRs can be delivered. We additionally show that, if RERRs must be sent unreliably, temporary routing-table loops caused by undelivered RERRs are broken within a finite time and data packets are never forwarded in a loop.

Although, we do not analyze the correctness of FLR with destination-sequenced labels, it directly follows from the correctness of LDR [8] because FLR uses feasible labels instead of feasible distances both of which share the same properties.

3.6.1 Loop-Freedom in FLR

Theorem 12. *In FLR, RREQs and RREPs do not loop.*

Proof. For a given route computation (A, ID_A), a node may be passive, engaged, or active. A node can become active in a route computation at most once, because it maintains the identifiers it assigns to the RREQs it originates. A router can engage in a route computation only when the the corresponding RREQ does not include the node in the path traversed by the RREQ. Hence, any RREQ can traverse only a directed acyclic graph (DAG), which may be a directed tree if no node relays the RREQ more than once, and any path traversed by a RREQ is free of loops.

Because RREPs are forwarded along the reverse path traversed by the corresponding RREQs, it follows that the RREPs must traverse loop-free paths. Furthermore, if the source route request (SRREQ) optimization is used, the SRREQ travels the loop-free path specified in the SRREQ by its source. □

Theorem 13. *FLR ensures that, if RERRs are sent reliably and path $P = \{n_k, ..., n_1\}$ exists at some point in time as defined by the successor entries of the nodes along the path, it is true that $FL_{n_1}^{n_i} > FL_{n_1}^{n_{i-1}}$, for $i \in [2, k]$.*

Proof. For path P to exist at a given time t, it must be true that all nodes in P have a valid successor. According to ALC, node n_i can make n_{i-1} its successor for destination n_1 if it received a RREP from n_{i-1} and either $L_{n_1}^{rep} < FL_{n_1}^{n_i}$ or n_i sent a RERR reliably to every node in its predecessor set, which forces all such nodes to stop using n_i as their successor for n_1. Hence, if P exists at time t, every node n_i ($i \in [2, k]$) must have accepted a RREP from n_{i-1} with $L_{n_1}^{rep} < FL_{n_1}^{n_i}$. Because $FL_{n_1}^{n_{i-1}} \leq L_{n_1}^{n_{i-1}} = L_{n_1}^{rep}$, the theorem is true. □

Theorem 14. *FLR is loop-free at every instant if RERRs are delivered reliably.*

Proof. Let node I initiate a RREP for destination D and let the RREP traverse the path P=$\{n_1,...,n_j\}$, where $n_1 = I$ (maybe the destination D) and $n_j = A$. Let the path from I to D be Q=$\{m_1, ..., m_k\}$, where m_1= D and $m_k = I$ and Q can be null if $n_1 = D$.

For a loop to form, node A must be on the path Q and must change successors after processing the RREP generated by I. From Theorem 12, path P must be loop free. Thereby, if $n_1 = D$, node A cannot form a loop after processing I's RREP. If $n_1 \neq D$ we must show that it is impossible for A to be on I's successor graph.

Assume that, at time t_0, path Q exists and is loop free and at time t_1 node A changes successors after processing I's RREP. Node I sends its RREP along the loop-free path P to A, carrying $L_D^{rep} = L_D^I(t_0)$. Let node A be some node m_i, $1 < i < k$ and let m_{i+1} be its predecessor. From Theorem 13 we have that $FL_D^I(t_0) > FL_D^A(t_0)$. Node A's routing-table entry for destination D can be in one of the following states.

Case (i): Node A has an invalidated route, which means A reliably notified its predecessor m_{i+1}, and the path Q no longer exists. Hence, node A cannot form a loop when it processes the RREP from I.

Case (ii): Node A has an active route. We know that $L_D^{rep} \geq L_D^I(t_0) \geq FL_D^I(t_0) > FL_D^A(t_0)$. At any time $t_0 \leq t \leq t_1$, node A's feasible label FL_D^A could have only decreased if A lies on path Q within this time interval. Hence, $FL_D^A(t_1) \leq FL_D^A(t_0)$. Now $L_D^{rep} \geq FL_D^A(t_1)$, so node A will not process the RREP, because it does not satisfy ALC. If A increased its feasible label when it chose a new successor, then it must have reliably notified m_{i+1} with a RERR, which reduces to case (i). Therefore, no loop cannot form in this case.

Case (iii): Node A has an invalidated route and is performing a local repair operation. As discussed for Case (ii), ALC is not satisfied at A when I's RREP is received. Although node A processes the RREP, it must send a RERR reliably to its predecessor set because $FL_D^A(t_1) \leq L_D^{rep}$. Hence, the path Q is no longer valid, because node m_{i+1} stops using node n_i as successor. Therefore, no loops can form in this case. □

We now prove that, if an undelivered RERR causes a loop after nodes reset their feasible label according to RLC or ALC, then data packets are never forwarded along the loop and the routing-table loop is broken within finite time.

Theorem 15. *FLR ensures that data packets never flow in loops and routing-table loops can exist only temporarily.*

Proof. A routing-table loop can form only when the ordering criteria of Theorem 13 is violated. This means that along a path $P = \{n_k, ..., n_1\}$ for destination n_1, $\exists i$, $i \in [2, k-1]$, $FL_{n_1}^{n_{i+1}} < FL_{n_1}^{n_i}$. Node n_i or node n_{i+1} can never update its routing table to create this condition, unless node n_i sent a RERR as per ALC or RLC before increasing its feasible label $FL_{n_k}^{n_i}$.

For the ordering violation to occur, n_{i+1} must not have received the RERR. Before increasing the feasible label at n_i, it removes n_{i+1} from the predecessor list. With the ordering criteria violated, a loop can be formed at a later time if node n_i's route request is replied to by a node upstream of n_{i+1} or node n_{i+1} itself. Assume such a loop is formed. If n_i receives a data packet from n_{i+1}, then node n_i must drop the data packet, and send a RERR for destination n_1 to n_{i+1}, because it received a data packet from a node that is not in the predecessor set for n_1. If node n_{i+1} receives the RERR, it invalidates its route to n_1 and the loop is terminated. On

the other hand, if n_{i+1} never sends a data packet, then the route entry expires and it no longer uses n_i as its next hop. Therefore, the routing-table loop terminates as both events are bounded by a finite-time duration, and data packets are never forwarded in loops even if the underlying routing table has loops. □

3.6.2 Correct Termination in FLR

We now prove that any source is able to establish a route to a destination within finite time if there is a physical path between the source and the destination, and the network is stable and error-free after an arbitrary sequence of topology changes. We assume that reliable RERRs can be delivered in the network. Also, we show that when a destination is partitioned from a set of nodes in a connected component, then all nodes in the connected component will invalidate their routes in finite time.

Lemma 1. *If an intermediate node I initiates a RREP for destination D using SLC in response to a RREQ from A that traversed a path P, then the concatenation of path P and the successor path from I to D is loop-free.*

Proof. Let the RREQ initiated by A traverse the path P=$\{n_1,... \ n_{k-1}\}$ before it is received by I. From Theorem 12, it follows that path P is loop-free. Assume that node I has a path Q=$\{m_1,...,m_{k-1}\}$ to destination D, which is its current successor path for the active route. We have to show that any node $n \in P$ is not the same as any $m \in Q$ for the lemma to be true. If Q is empty, then the lemma is trivially true.

The proof for the case where Q is not empty is by contradiction. When the RREQ is received by node I, it carries the minimum of the feasible labels (MFL_D^{req}) of the nodes along

the path P. Because I has an active route to D, by Theorem 13, we have along path Q from I to D that $FL_D^I > FL_D^{m_1} > FL_D^{m_2} > ... > FL_D^{m_{k-1}} > FL_D^D$. At any instant, we have $L_D^I \geq FL_D^I$. For the RREQ to satisfy SLC at node I, we must have $L_D^I < MFL_D^{req}$. However, if any node $m \in Q$ is the same as any node $n \in P$, we will have $L_D^I > MFL_D^{req}$ and node I cannot initiate the RREP satisfying SLC. Therefore, the concatenation of path P and successor path of I must represent a loop-free path to the destination. □

Theorem 16. *In an error-free stable connected network, FLR ensures that a node A starting a route computation (A, ID_A) for a destination D establishes a successor path to the destination within a finite time.*

Proof. We consider the first RREP rp_A that reaches A for the route computation (A, ID_A). If multiple RREPs are received, then node A can switch to better routes (i.e., routes with smaller labels).

Let node A start a new route computation (A, ID_A) for destination D, by sending a RREQ, and let that RREQ traverse the path $P = \{n_1..n_{k-1}\}$ before arriving at node n_k (which can be D) which satisfies SLC. From Lemma 1, if nodes switch along path P concatenated with the successor path from n_{k-1}, they have a loop-free valid successor path to the destination.

Let node n_k generate a RREP to the RREQ from source A. The proof must show that when nodes relay the reply along the reverse path, nodes that are engaged in the computation (A, ID_A) after have a valid successor path to the destination after processing the reply.

We first prove that A establishes the successor path using the RREP relayed along path P when no node along the path P is affected by another route discovery event for D during the

route computation (A, ID_A). In this case, no node along P satisfies SLC with A's feasible label, for otherwise that node would have responded to the RREQ instead of n_k. Each node $n \in P$ must be in one of three states: (i) n has no information about D, (ii) n's information is invalid, or (iii) n has an active route but SLC could not be satisfied. In Case (i), node n may use any RREP sent by n_k. In Case (ii), node n has an invalid route, which means that n has notified any predecessors reliably through a RERR and node n can process the RREP and update its route entry for D. In Case (iii), node n can process the RREP and switch successors if the reply offers a smaller label; if node n does not switch successors, then it has a better route to D. In all three cases, node n will then forward a new RREP with its current label and forward it along the reverse path carried in the RREP. Node A will receive the RREP, because nodes in all states forward a RREP along the reverse path of P.

We now show that simultaneous route discovery events for destination D do not affect the route computation (A, ID_A) and nodes along path P relay a RREP to A. The following three cases represent the events that can interfere with the route computation of (A, ID_A).

Case 1: During the computation period of (A, ID_A), one or more nodes $n_i \in P$ are engaged in route computations (m_i, ID_{m_i}) for destination D, where $m_i \notin P$. Denote by rp_A the RREP initiated by route computation (A, ID_A) and by rp_m the set of RREPs for (m_i, ID_{m_i}). From the previous discussion, node n_i on processing rp_A or any rp_m, may update its routing table and forward a new RREP along the reverse path. If node n_i receives rp_A before any rp_m, the route computation (A, ID_A) has completed at node n_i. On the other hand, if rp_A was received after a $rp \in rp_m$, then this reduces to case (iii) above, when there are no other simultaneous route discovery events and node n_i may improve its route after processing rp_A. In

either sequence of events, node n_i forwards a new RREP with its current label along the reverse path to A, which ensures that the route computation (A, ID_A) is not affected by actions at n_i.

Case 2: Nodes $m_i \in P$ become active during the route computation period of (A, ID_A). A node m_i becoming active for destination D is equivalent to the node being engaged in a new route computation for D. This reduces to Case 1 and the route computation for (m_i, ID_{m_i}) cannot interfere with that of (A, ID_A).

Case 3: One or more nodes $m_i \in P$ are engaged and one or more nodes $m_j \in P$ are active for route computations to destination D. From Case 2, nodes that are active for a route computation to destination D reduce to the case of being engaged in the route computation. Hence, we can consider $m_j \in P$ to be engaged. So from Case 1, the route computations cannot interfere with that of (A, ID_A).

Therefore, the nodes along path P engaged in the route computation $(A,\ ID_A)$ are not affected by simultaneous route discovery events for destination D. Node A receives the RREP forwarded along the reverse path of P and establishes a successor path to the destination within a finite time given that each message is exchanged within a finite time. □

Theorem 17. *In a connected component, all nodes partitioned from a destination will invalidate their routing entries for that destination within a finite time.*

Proof. Let G_D be a set of nodes that belong to a connected component partitioned from D. Assume node D is inaccessible to the nodes in G_D after time t. Consider the case in which a node $i \in G_D$ never invalidates its route after a time $t_i > t$. Because FLR is loop-free at every instant, after a finite time $t_f \geq t$, all paths in G_D must lead to nodes that have invalidated their

routes (set $L_D = \infty$). And, these nodes must have sent a RERR to their predecessors on this path and it must eventually be received by node i. Therefore, for node i to have a valid route at time t_f, it must re-learn its route by receiving a RREP from a neighbor that satisfies ALC, and this must happen in succession for an infinite number of times after t. Assuming a stable topology after time t, when any node $m \in G_D$ joins the directed acyclic successor graph for D by assigning itself a label containing i (i.e., $L_D^m \ni .. \oplus i \oplus .. \oplus D$), node i cannot accept new RREPs from m because ALC cannot be satisfied (i.e., $i \in L_D^m$). Therefore, for node i to be able to re-learn a route for D an infinite number times in succession after time t, there must exist an infinite number of paths (labels) from node i to j which is not possible because G_D has a finite number of nodes. Hence, every node $i \in G_D$ will invalidate its route a finite time after time t. □

3.7 Performance Comparison

We present results for FLR (without any optimizations) over varying loads and mobility. The protocols used for our comparison with FLR are DSR, AODV, and OLSR, which are representative of the state of the art in routing protocols for MANETs. We also simulate two variants of FLR (from Section 3.3.4.5) : (i) FLR-DL (FLR with destination-sequenced labels based on Approach 2), (ii) FLR-DL-LR (FLR-DL with the one-hop local repair scheme). To validate our claims that performance can be improved with destination-sequenced labels and be applied to current MANET routing protocols, we simulate AODV-PA (AODV with path accumulation). We evaluate the performance of our local repair scheme by comparing FLR-DL-

Table 3.2: Performance average over all pause times for 50 nodes network for 10-flows and 30-flows (random)

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
FLR	10	0.994±0.002	0.019±0.003	0.256±0.060	2.500±0.171
FLR-DL	10	0.980±0.076	0.022±0.004	0.350±0.099	2.450±0.258
FLR-DL-LR	10	0.994±0.002	0.023±0.004	0.381±0.100	2.487±0.182
AODV	10	0.994±0.002	0.016±0.003	0.270±0.066	2.576±0.179
AODV-LR	10	0.994±0.002	0.017±0.004	0.266±0.067	2.580±0.180
AODV-PA	10	0.994±0.002	0.017±0.005	0.268±0.065	2.583±0.190
DSR	10	0.940±0.027	0.041±0.047	0.220±0.095	2.677±0.185
OLSR	10	0.887±0.040	0.012±0.001	1.937±0.220	2.456±0.175
FLR	30	0.820±0.051	0.491±0.180	2.005±0.722	2.717±0.268
FLR-DL	30	0.845±0.049	0.582±0.232	2.456±0.819	2.744±0.254
FLR-DL-LR	30	0.865±0.055	0.518±0.259	1.991±0.828	2.776±0.270
AODV	30	0.765±0.055	1.010±0.356	4.423±1.289	2.951±0.324
AODV-LR	30	0.770±0.056	0.965±0.333	4.269±1.264	2.929±0.309
AODV-PA	30	0.754±0.055	1.092±0.351	4.343±1.191	3.002±0.336
DSR	30	0.683±0.059	4.760±1.073	0.410±0.140	3.625±0.308
OLSR	30	0.798±0.034	0.883±0.311	0.713±0.069	2.478±0.161

Table 3.3: Performance average over all pause times for 100 nodes network for 10-flows and 30-flows (random)

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
FLR	10	0.989±0.004	0.043±0.007	0.838±0.251	3.671±0.340
FLR-DL	10	0.988±0.004	0.064±0.012	1.541±0.440	3.634±0.308
FLR-DL-LR	10	0.987±0.004	0.062±0.012	1.452±0.420	3.650±0.307
AODV	10	0.988±0.004	0.036±0.009	0.897±0.236	3.744±0.293
AODV-LR	10	0.988±0.004	0.035±0.008	0.872±0.221	3.767±0.293
AODV-PA	10	0.988±0.004	1.682±0.445	0.897±0.238	3.803±0.311
DSR	10	0.876±0.050	0.099±0.057	0.859±0.353	4.257±0.317
OLSR	10	0.821±0.063	0.022±0.002	11.795±1.575	3.583±0.256
FLR	30	0.648±0.047	0.874±0.188	8.347±1.795	4.355±0.370
FLR-DL	30	0.699±0.059	0.966±0.227	9.492±2.096	4.261±0.340
FLR-DL-LR	30	0.706±0.040	0.925±0.194	8.324±1.537	4.290±0.346
AODV	30	0.608±0.051	1.455±0.385	18.298±13.069	4.751±0.434
AODV-LR	30	0.592±0.044	1.617±0.538	21.339±15.523	4.868±0.463
AODV-PA	30	0.582±0.052	1.682±0.445	16.889±7.394	4.918±0.417
DSR	30	0.618±0.049	5.125±0.782	1.243±0.405	6.141±0.499
OLSR	30	0.612±0.041	3.371±0.532	5.423±0.669	4.014±0.277

Table 3.4: Performance average over all pause times for 50-nodes and 100-nodes network with 30-flows (fixed)

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
FLR	50	0.758±0.089	0.584±0.342	2.858±1.313	2.903±0.386
FLR-DL	50	0.803±0.070	0.629±0.357	3.153±1.160	2.864±0.345
FLR-DL-LR	50	0.803±0.072	0.622±0.354	2.918±1.166	2.933±0.401
AODV	50	0.738±0.083	0.866±0.473	5.044±1.874	3.084±0.424
AODV-LR	50	0.735±0.082	0.865±0.457	5.095±1.861	3.095±0.420
AODV-PA	50	0.727±0.082	0.963±0.505	4.942±1.739	3.149±0.441
DSR	50	0.581±0.081	3.422±0.813	0.430±0.156	3.809±0.406
OLSR	50	0.772±0.042	0.842±0.413	0.727±0.072	2.534±0.187
FLR	100	0.623±0.100	0.735±0.314	9.307±3.622	4.504±0.707
FLR-DL	100	0.682±0.074	0.818±0.296	9.976±2.763	4.338±0.557
FLR-DL-LR	100	0.707±0.074	0.729±0.278	8.143±2.439	4.352±0.548
AODV	100	0.608±0.088	1.060±0.402	16.812±8.743	4.731±0.696
AODV-LR	100	0.595±0.094	1.133±0.440	19.050±12.564	4.882±0.738
AODV-PA	100	0.572±0.085	1.322±0.480	16.679±6.446	4.957±0.695
DSR	100	0.476±0.099	3.865±1.150	1.208±0.453	6.177±0.733
OLSR	100	0.585±0.066	2.761±1.062	5.541±0.811	4.074±0.452

LR against AODV enhanced with local repair [19]. We could not simulate DSR's local repair scheme because it requires a pre-defined salvage count to break loops, which is not specified in the draft. Simulations are run in Qualnet[44]. The parameters are set as Perkins et al describe in their work [41].

3.7.1 Simulation Setup

Simulations were performed on two scenarios, a 50-node network with terrain dimensions of 1500m x 300m, and a 100-node network with terrain dimensions of 2200m x 600m. Traffic loads were CBR sources with a data packet size of 512 bytes. Load was varied by using 10 flows (at 4 packets per second) and 30 flows (at 4 packets per second). We use two sets of traffic characteristics: (i) random flows have a mean length of 100 seconds, distributed exponentially, and (ii) fixed flows that last the entire simulation time. For the fixed flows, we only

show results for 30-flows because we did not notice any shift in trend from the one observed for the 10-flows scenario with exponential flow distribution. The MAC layer used was IEEE 802.11 with a transmission range of 275m and throughput 2 Mbps. The simulated time is 900 seconds. Node velocity was set between 1 m/s and 20 m/s. Flows have a an exponentially distributed length with a mean of 100 seconds. Each combination (number of nodes, traffic flows, scenario, routing protocol and pause time) was repeated for nine (9) trials using different random seeds.

3.7.2 Performance Metrics

We address four performance metrics. *Delivery ratio* is the ratio of the packets delivered per client/server CBR flow. *Latency* is the end to end delay measured for the data packets reaching the server from the client. The *network load* is the total number of control packets divided by the number of received data packets. *Data hops* is the number of hops traversed by each data packet (including initiating and forwarding) divided by the total number of received packets in the network. This metric takes into account packets dropped due to forwarding along incorrect paths, and provides a measure of the quality of the routes.

3.7.3 Performance Discussion

Tables 3.2 and 3.3 summarize the results of the different metrics by averaging over all pause times for the 50 and 100 node networks with random exponentially distributed flows. Table 3.4 summarizes the same set of metrics for 50 and 100-node networks with fixed flow distributions. The columns show the mean value and 95% confidence interval. All our performance discussions focus on the average case since the confidence intervals overlap atleast

slightly in most cases. The packet delivery ratio, the end-to-end delay, and the control overhead over various pause times for 50-node and 100-node networks with 30-flows is shown for random exponential flows in Figures 3.5 and 3.3, and for fixed flows in Figures 3.6 and 3.4. The vertical bars in the graphs indicate the 95% confidence intervals.

From the summarized results, in the 10-flow scenarios (exponentially distributed traffic), the performance metrics for the FLR and AODV variants do not show any notable difference. However, both DSR and OLSR exhibit poor packet delivery, which indicates that corrupted topology information due to mobility affects the accuracy of the routes in both the protocols even at very light-loads. The problem with DSR lies in the use of source-routes to deliver packets. Source-routes are invalidated due to topology changes and data packets are forced to be dropped by intermediate nodes. This problem is aggravated by the use of an optimization to learn source-routes carried in data packets that can be stale. OLSR's performance suffers from the temporary loops that can exist until the latest topology change is disseminated to all nodes in the network. The loops cause unwanted transmission of data packets, thereby congesting the physical medium which in-turn affects the dissemination of topology updates.

The results in the 30-flow scenarios show much more disparity in the average performance of the protocols although most results lie in overlapping confidence intervals. The packet delivery graphs across various mobility pause times (Figs. 3.6(a), 3.5(a), 3.4(a), 3.3(a)) show that FLR has a very consistent performance, and performs far better in high-mobility scenarios. The reason for the performance can be attributed to more intermediate replies. This serves two purposes: Routes are recovered more quickly, and costly RREQ floods to the destination, which in turn can cause more congestion, are avoided. The performance of AODV

suffers in these scenarios, where RREQs mostly get answered only by the destination. This is further aggravated by the congestion caused by these floods. OLSR, and DSR benefit from very low-mobility scenarios where the topology information stays more accurate. OLSR performs more consistently than DSR and it has a slightly higher average packet delivery than FLR in the 50-nodes, 30-fixed flows scenario. However, this is largely due to its better performance under low-mobility. FLR and its variants exhibit the lowest latency characteristics across all scenarios. The control overhead of FLR and its variants are lesser than that of AODV and its variants. Although, OLSR and DSR have smaller control overhead than FLR and AODV, they cannot be directly compared. The reason is that OLSR's control overhead is independent of the traffic flow characteristics, and DSR floods fewer RREQs because of the optimization through which it can learn source-routes carried in data packets. A higher control overhead usually results in congesting the medium, which in turn causes delay when transmitting data packets. The low delay characteristics of FLR can attributed to its control overhead which is lesser than AODV. Despite the low control-overhead, the delays in OLSR are caused by looping of data packets and buffering of data packets in DSR as delivery using new source-routes are being attempted.

The termination issues associated with AODV and its variants can be seen in the performance of the 100-nodes scenario with 30-flows. We were unable to obtain tight confidence intervals for the control overhead of AODV in the summarized results. We believe that the reason for the excessive control overhead and poor packet delivery (Figs. 3.4 and 3.3) in the scenarios with very high mobility are due to the counting-to-infinity and temporary loops [15] [14] that can occur in AODV when there is heavy congestion of the wireless medium. FLR and its variants do not display any of this behavior. In these cases, the FLR variants with

destination-sequenced labels have better packet delivery and tighter confidence intervals than FLR, because of the absence of temporary loops, which FLR blocks but does not prevent with an unreliable MAC.

More interesting to note is the relative performance between FLR and its variants which highlights their specific properties. The two FLR variants based on destination-sequenced labels exhibit a better average performance than FLR, and this difference is much more significant in the heavy-load (30-flows), 100-nodes scenarios. This shows that route errors might not be delivered reliably with increasing congestion and there is more chance for the ordering of the labels to be violated in FLR, thereby, resulting in temporary loops or bad routes. This situation does not occur when destination-sequenced labels are used because the RREQs are relayed with a 'reset' and is answered by the destination with a higher sequence number. This effect is also noticeable in the *increased* control overhead and delay of the DL-based variants compared to FLR. The results show the trade-offs between operating FLR based on packet-filtering in networks with low-congestion, where the performance is almost equivalent to using DL's; however, or in large networks with heavy-loads, where the performance of FLR is affected due to temporary loops. This trade-off is also evident because FLR has a lower control overhead and latency compared to its variants indicating that the lack of 'resets' are beneficial when downstream intermediate nodes reply correctly, but can be bad when upstream nodes answer incorrectly.

AODV with path accumulation (AODV-PA) performs about the same as AODV. This is due to the fact that the optimization allows route entries to be setup for relay nodes, and this will only improve performance when there are flows towards the relay nodes within the small window of time before the routes expire. These results are in direct correlation with the

performance results shown in [11], where it was found to show improvements only in huge networks with many flows and was proposed as an optimization in such a case. Given that FLR-DL requires the same signalling as AODV-PA, its performance demonstrates that using path information as the basis for labeling from which ordering is derived is much more effective than simply attempting to learn routes to relay nodes and discarding the path information.

The local repair scheme of FLR-DL-LR shows a noticeable improvement over FLR-DL, with improvements in packet delivery across most scenarios and reduced latency and control overhead more closer to that of FLR. The simple one-hop query allows intermediate nodes to find an alternate path without requiring to inform the source which then will start a new RREQ flood. This scheme is much more effective than the local repair scheme of AODV, which shows no difference from that of AODV. In FLR-DL-LR, the intermediate node either finds a neighbor that has a lower cost label with the same sequence number or a higher sequence number, or fails to repair. In contrast, the AODV local repair scheme performs a ttl-controlled RREQ flood to search for the destination in the vicinity, and this is further limited by checks required before performing a local-repair, such as knowing the hop-count to the source.

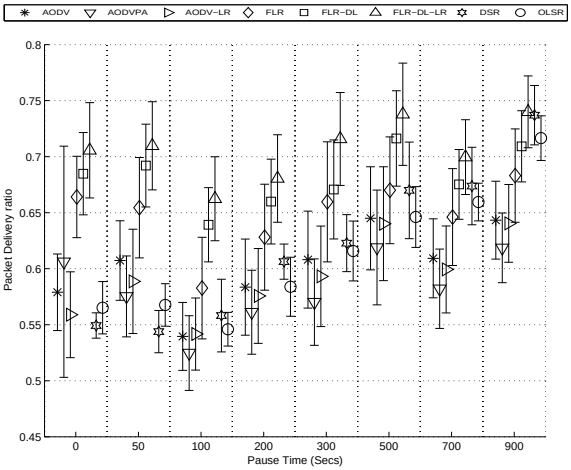
The *data hops* metric provides a measure of the accuracy of the routes used for forwarding. The data hops metric reflects the number of hops traversed by each data packet whether or not it is delivered. By correlating the packet delivery ratio with the data hop count, a notion of how many packets were actually delivered to the destination can be gauged. All protocols have statistically equivalent data hops across all scenarios, which means that in protocols delivering less packets, some of the packets are dropped at the intermediate nodes. In AODV, packets are dropped at intermediate nodes due to falsely triggered route failures or temporary

loops with heavy congestion. Data packets in OLSR can temporarily traverse loops before being delivered or can get dropped due to lack of routes. Stale source routes in DSR, due to mobility, cause packets to be dropped at the intermediate nodes. The data hops of FLR and its variants in correlation with the packet delivery ratio shows the high accuracy of the active routes, and FLR-DL's route accuracy is better than FLR due to instantaneously loop-free routing tables.

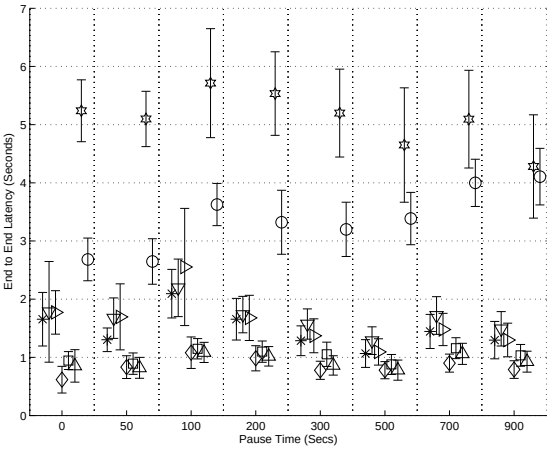
3.8 Conclusions

We extended sufficient conditions for loop-free routing previously stated for distances to labels that are ordered lexicographically. We introduced the feasible label routing (FLR) protocol as an illustration of how loop-free routing can be attained using path information on demand while allowing data-packet forwarding on the basis of the packet destinations only.

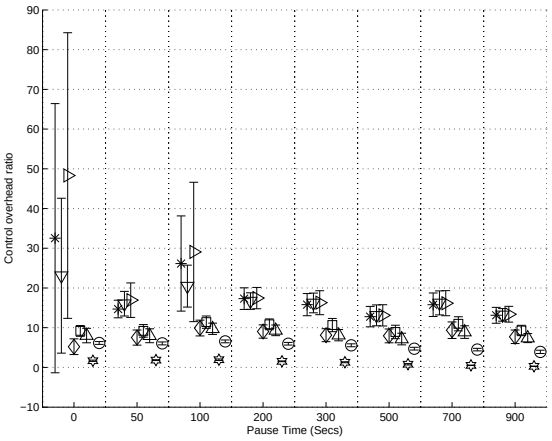
In FLR, nodes are ordered according to their labels to a destination to provide instantaneous loop-freedom. A node resets its label (path) for a destination when its route fails. Reseting labels requires a reliable route error message to predecessors. To cope with the case in which the MAC layer of a MANET does not support reliable transmissions efficiently, we proposed using a local data-packet filtering action to detect and break routing-table loops caused when RERRs are sent unreliably. Destination-sequenced labels created by associating per-destination sequence numbers with labels can be used in FLR to maintain instantaneously loop-free routing tables. Simulation results show that FLR and variants based on destination-sequenced labels outperform DSR, AODV, AODV-PA (AODV with path accumulation), and OLSR.



(a) Packet Delivery

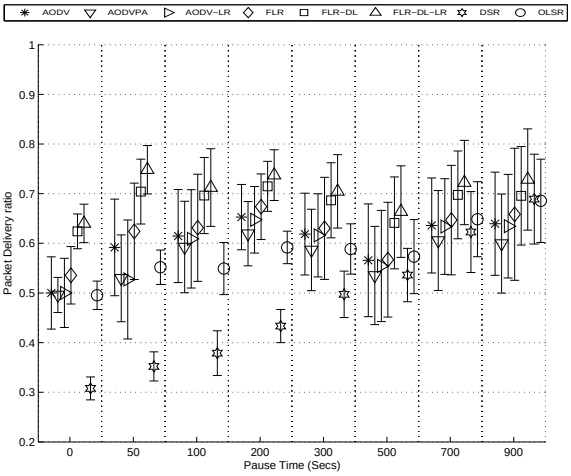


(b) Latency

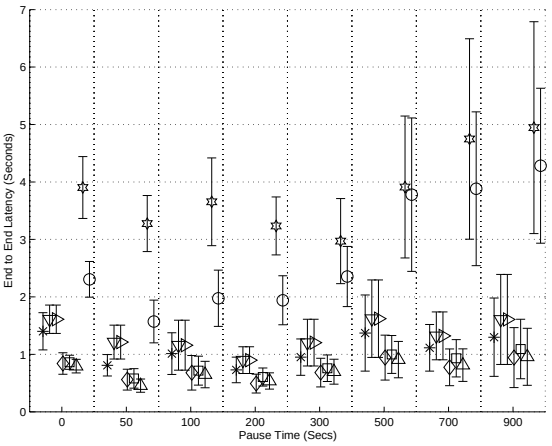


(c) Control Overhead

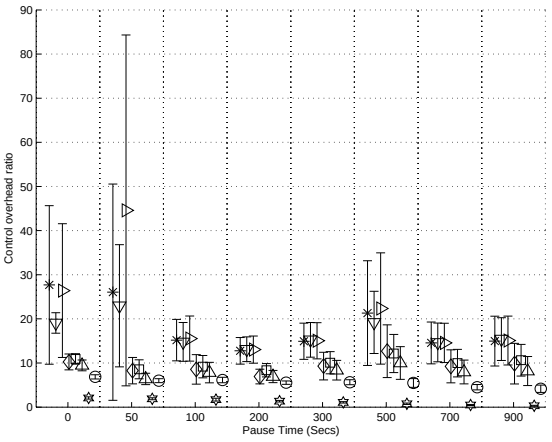
Figure 3.3: Random (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

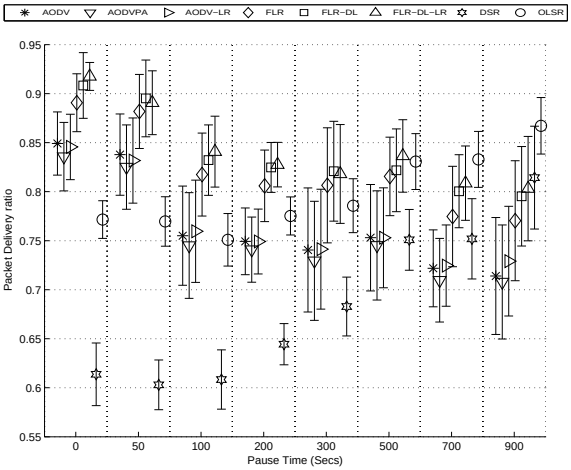


(b) Latency

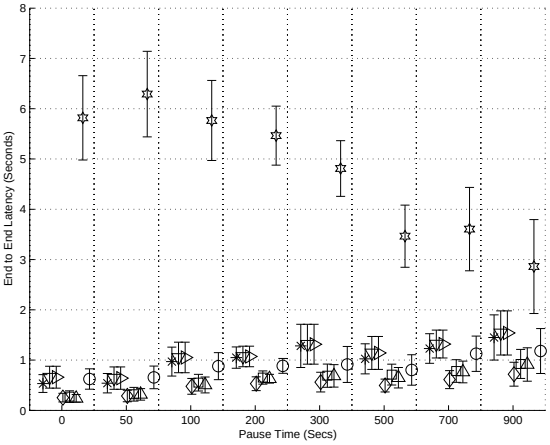


(c) Control Overhead

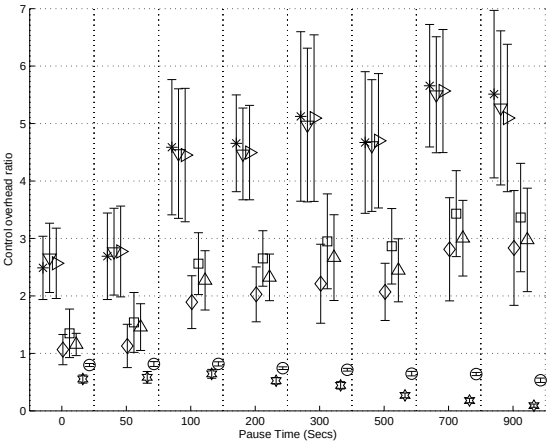
Figure 3.4: Fixed (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

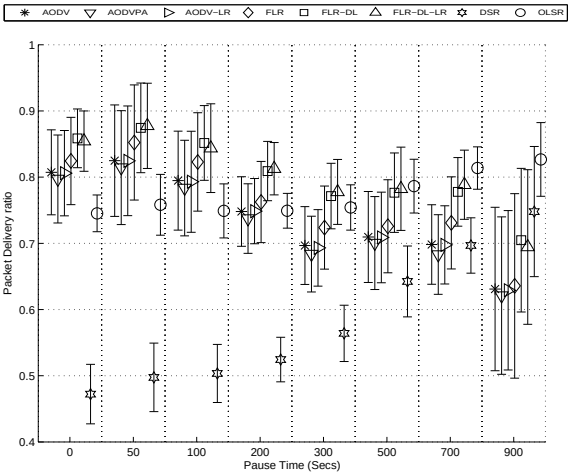


(b) Latency

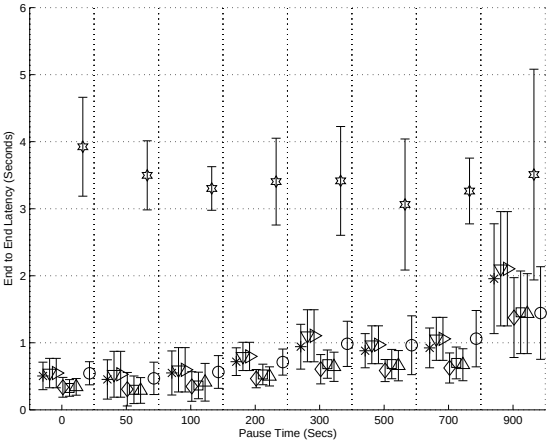


(c) Control Overhead

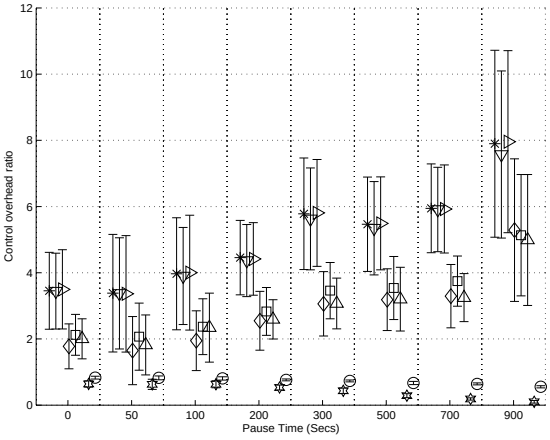
Figure 3.5: Random (50-nodes, 30-flows, 120 pps)



(a) Packet Delivery



(b) Latency



(c) Control Overhead

Figure 3.6: Fixed (50-nodes, 30-flows, 120 pps)

Chapter 4

Routing using trusted topology information

4.1 Introduction

Although a variety of on-demand routing protocols based on topology information have been proposed, none of these schemes use topology information as the basis to ensure that loop-free routing tables are maintained, and, also, no solution exists that provides instantaneous loop-free routing without requiring source-routed packets, data-packet filtering, or reliable information exchanges among neighbors. In this chapter, we introduce the Link Vector Routing (LVR) protocol for loop-free on-demand routing in ad hoc networks. LVR differs from all prior routing protocols based on topology information in that nodes communicate and store information about those links that should not be trusted when rerouting decisions are made, in addition to the information about known links in the network. Section 4.2 presents and proves a sufficient condition for loop-freedom using link vectors specifying the paths nodes use to reach destinations and the set of links that caused route changes and cannot be trusted. Section 4.3

details the principles of operation and control signaling of LVR, which is remarkably simple: As in prior schemes, routers use RREQs to discover valid path information and RREPs to provide the requested path information to sources of data packets. However, LVR also uses RREQs and RREPs to disseminate information about those links that should not be trusted and should not be considered in repairing routes to destinations between a node requiring a new route to the destination and the destination itself. A node can send a RREP only if its active path to the requested destination does not contain any link traversed by the RREQ or any link specified by the source and the relays of the RREQ as one that cannot be trusted. This dissemination notifies nodes about invalid topology information and ensures that intermediate nodes sending RREPs do so considering only correct topology information. Section 4.4 provides an example of how LVR works, and compares it to DSR's packet salvaging technique that can result in loops. Section 4.5 analyzes the correctness of LVR.

Section 4.6 presents the results of simulations run in Qualnet for a number of scenarios comparing LVR with OLSR, AODV, DSR, FLR, and AODV-PA. The results clearly indicate that LVR performs better than the other protocols. This is due to the fact that LVR allows more intermediate nodes to answer RREPs using paths based on *valid* topology information, because RREQs and RREPs specify those links that caused prior RREQs to be issued. Additionally, LVR uses the topology information cached at nodes to send *source-routed* RREQs similar to those in OLIVE and the flow establishment packets in flow-based DSR [12] to reduce the need for flooding RREQs. Section 4.7 presents our concluding remarks.

4.2 Sufficient conditions for Loop-Freedom using link vectors

4.2.1 Network and routing Model

To describe our loop-free condition, we refer to those nodes that use a given node in their paths to a destination as "upstream nodes" of the given node.

The topology of the network is considered as a directed graph $G = (V, E)$, where V is the set of nodes and E is set of links connecting the nodes. Each edge $e \in E$ has an associated time-varying link cost *lc*. A link $e \in E$ represents a directed arc (u, v), where $u, v \in V$; node u is called the *head* of link e, and node v is called the *tail* of link e. The sequence of links, which we call link vector, $LV = \{e_1, e_2, ..., e_n\}$ (where $e_i \in E$ with $i \in \{1, ..., n\}$) represents a directed path from the head of link e_1 to the tail of link e_n.

The following information is stored at node A for a destination D: (a) the successor or next hop (s_D^A), (b) a link vector (LV_D^A) consisting of the last known valid path from A to D, and (c) the link coordination set (LC_D^A) consisting of an unordered collection of links. The link vector LV_D^A need not be accurate at every instant of time due to topological changes downstream. The link coordination set LC_D^A contains those links that A should not be use when choosing new paths for destination D.

An update message for destination D generated by node A consists of a link vector LV_D^A, and the link coordination set LC_D^A, stored by A at that time. The most recent link vector for destination D received from node B by node A is denoted by LV_{DB}^A. The operator $\oplus$ is used to add a new link in order to the chain of nodes in the link vector.

Node A applies the following two rules when it must change its successor for desti-

nation D after an input event:

- *R*ule 1: If node A must make node B its successor for destination D, it sets $LV_D^A \leftarrow (A,B) \oplus LV_{DB}^A$.

- *R*ule 2: If node A must change its path to D because its link to its current successor s_D^A fails, then node A sets $LC_D^A \leftarrow LC_D^A \cup (A, s_D^A)$ before it changes s_D^A.

4.2.2 Loop-free condition

Link Vector Condition (LVC): Node A can make node B its new successor for destination D at time t if (i) $\nexists\, l \mid (l \in LC_D^A(t) \ \wedge \ l \in LV_{DB}^A(t))$, and (ii) $LC_{DB}^A(t) \supseteq LC_D^A(t)$. If no such neighbor B exists, then node A must retain its current successor for D, if it has any.

The first part of the condition states that node A can pick neighbors advertising a path that does not contain a link that should not be trusted. The second part implies that the nodes downstream a chosen path to D do not trust sets of links that include all the links that node A does not trust.

Theorem 18. *No routing table loops can form for destination D if nodes use LVC whenever they change their successor paths for destination D.*

Proof. The proof is by contradiction. Without loss of generality, assume that the directed successor graph for destination D, which we denote by $S_D(G)$, is loop-free at every instant before time t. Assume that a loop $L_D(G)$ is formed at time t. It is easy to see that no loops can be formed unless at least one node picks as its successor at time t a node that is upstream in $S_D(G)$.

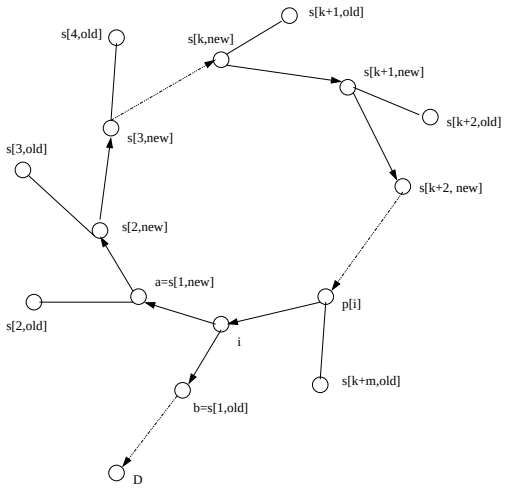


Figure 4.1: LVC loop-free condition

Assume that $L_D(t)$ is formed when node i makes node a its new successor $s_D^i(t)$ after processing an input event at time t, where $b = s_D^i(t_b) \neq a$ and $t_b < t$. We denote by $P_{aj}(t)$ the path formed by the successor entries of the nodes starting from node a to node j for destination D. Because we focus on shortest-path routing, $P_{aD}(t)$ must include $P_{ai}(t)$.

Let $P_{ai}(t)$ consist of the chain of nodes {a= s[1,new], s[2,new],...,s[k,new],...,i} at time t, as shown in Figure 4.1. The notation indicates that node s[k+1, new] is the k^{th} hop in path $P_{ai}(t)$ and has node s[k+1,new] as its successor at time t.

The last time that node s[k,new] updates its routing table entry up to time t and sets $s_D^{s[k,new]}$=s[k+1,new] is denoted by $t_{s[k+1,new]}$, where $t_{s[k+1,new]} \leq t$. Therefore, we have that $s_D^{s[k,new]}(t_{s[k+1,new]}) = s_D^{s[k,new]}(t)$.

Because nodes joining P_{aD} do not switch to any new successors afterwords, it is also true that

$$LC_D^{s[k,new]}(t_{s[k+1,new]}) = LC_D^{s[k,new]}(t) \tag{4.1}$$

The time when node s[k,new] sends an update that constitutes the last update from such a node that is processed by node s[k-1, new] up-to time t is denoted by $t_{s[k+1,old]}$. Node s[k, new]'s successor at time $t_{s[k+1,old]}$ is denoted by s[k+1, old]. Note that $t_{s[k+1,old]} \leq t_{s[k+1,new]} \leq t$, and that $s[k+1, old]$ need not be the same as $s[k+1, new]$.

Note that $LC_D^i(t_1) \subseteq LC_D^i(t_2)$, if $t_2 > t_1$. Because LVC must be satisfied when node s[k,new] $\in P_{aD}(t)$ makes node s[k+1,new] $\in P_{aD}(t)$ its successor at time $t_{s[k+1,new]}$ it must be true that

$$LC_{Ds[k+1,new]}^{s[k,new]}(t) = LC_{Ds[k+1,new]}^{s[k,new]}(t_{s[k+1,new]}) \supseteq LC_D^{s[k,new]}(t_{s[k+1,new]})$$

For a loop to be formed after time t, it must be true that $P_{aD}(t)$ exists. Hence, given that each node along $P_{aD}(t)$ must satisfy LVC when switching successors, we have that

$$LC_D^a(t) = LC_D^a(t_{s[2,new]}) \subseteq LC_{Ds[2,new]}^a(t)$$

$$= LC_D^{s[2,new]}(t_{s[3,old]}) \subseteq LC_D^{s[2,new]}(t_{s[3,new]})$$

...

$$LC_D^{s[k,new]}(t_{s[k+1,new]}) \subseteq LC_{Ds[k+1,new]}^{s[k,new]}(t)$$

$$= LC_D^{s[k+1,new]}(t_{s[k+1,old]}) \subseteq LC_D^{s[k+1,new]}(t_{s[k+1,new]})$$

...

$$LC_D^{p[i]}(t_i) \subseteq LC_{Di}^{p[i]}(t) = LC_D^i(t_{s[k+m,old]}) \subseteq LC_D^i(t_i)$$

... (4.2)

Accordingly, from Eqs. 4.2 and 4.1, it is true that

$$LC_D^a(t) \subseteq LC_D^{s[2,new]}(t) \subseteq ... \subseteq LC_D^{s[k,new]}(t)$$

$$\subseteq ... \subseteq LC_D^i(t) \subseteq ... \quad (4.3)$$

According to Rule 2, before node a switches to $s[2, new]$ at time $t_{s[2,new]}$, it must be true that $(a, s[2, old]) \in LC_D^a(t_{s[2,new]})$. Hence, $(a, s[2, old]) \in LC_D^i(t)$. Similarly for every node s[k, new] $\in P_{ai}(t)$, it is true that (s[k, new], s[k+1, old]) $\in LC_D^{s[k,new]}(t)$.

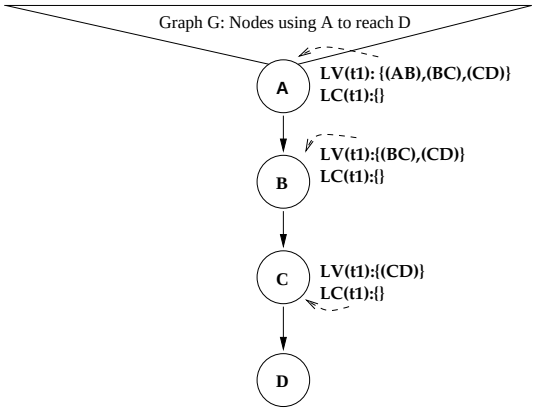
For a loop to be formed at time t, LVC must be satisfied for node i to switch to node a as successor, i.e., $\nexists l \mid (LV_{Da}^i(t) \ \wedge \ l \in LC_D^i(t)$. Depending on the last topological update processed by node a and received by node i, $LV_{Da}^i(t)$ can take different possibilities. Given that $LV_{Da}^i(t) = LV_D^a(t_{s[2,old]})$, if $LV_D^a(t_{s[2,old]}$ reflects the accurate link vector P_{aD}, then from Rule 2 it must be the case that $(i, b) \in LC_D^i(t)$. Therefore, LVC cannot be satisfied at node i in this

case. On the other hand, if $LV_D^a(t_{s[2,old]})$ is outdated and contains at least one of the previously used links (i.e., ($s[k, new]$, $s[k+1, old]$)) before a node $s[k, new]$ switched to path P_{aD}, then LVC is not satisfied at node i; hence, no loops can be formed.

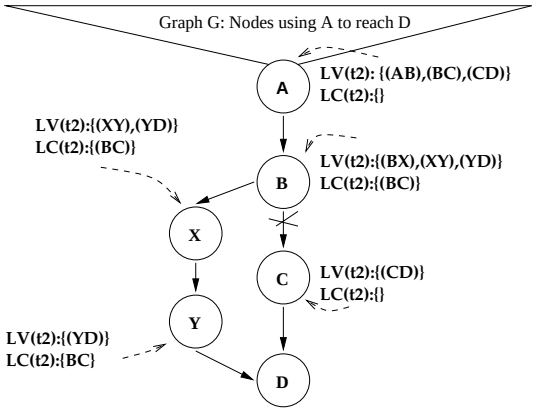
□

4.2.3 Illustration of loop-free condition

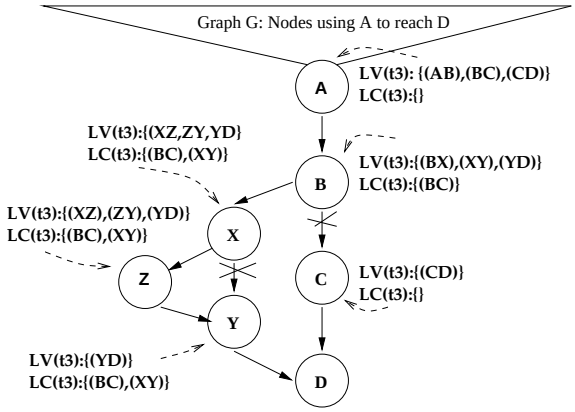
We illustrate how LVC can be used to ensure instantaneous loop-freedom assuming the existence of a mechanism that *coordinates* the link coordination sets along a loop-free path. Figure 4.2 shows the directed acyclic successor graph for a seven-node network with a flow from node A to node D. The corresponding *LV* and *LC* are shown with respect to their times *(t1, t2, t3)*. Assume that link BC fails at time $t1 < t < t2$. If node X had a route to D along, say, path P={X,Y,D}, then B can detect from $LV_D^X(t_1)$ that it is a loop-free path. However, before node B can switch to X at time $t2$, the second part of LVC must be satisfied along path P. Figure 4.2(b) shows the resulting LC's at time $t2$ after a mechanism was used to coordinate the LCs along the path. The key advantage of this coordination of LCs along the successor path P is that the update for the link failure (B, C) need not be reliably communicated to the entire set of nodes upstream of node A and the nodes in subgraph G. If node X tries to use A or any upstream node after time t_2, $LV_D^A = \{(AB), (BC), (CD)\}$ is detected to be out-dated because of $LC_D^X = \{(BC)\}$ (i.e., the LVC condition is violated); therefore, a loop cannot be formed. Again, if link XY fails at a time $t2 < t < t3$. Then, node X can switch to the path $P' = (XZ), (ZY), (YD)$ after the necessary link coordination, as shown at time t_3 in the figure. A new node downstream , say Y, cannot switch to node A despite the out-dated LV_D^A



(a) Time t1



(b) Time t2



(c) Time t3

Figure 4.2: LVC Illustration

due to the untrusted links stored in LC_D^Y.

The above example illustrates that loop freedom in a routing protocol based on LVC is achieved by requiring nodes to store in their link coordination set the set of links that should not be used in computing new routes. Hence, the nodes upstream of node B (node A, and graph G in the example) can be updated to the correct link vectors without affecting loop-freedom. This is true even if the updates propagating upstream are lost or delayed.

As Theorem 18 shows, LVC is sufficient to guarantee routing-table loop-freedom. However, a routing protocol based on LVC must consider the following issues: (i) When no neighbors satisfying LVC can be found and physical paths to the destination exist, a mechanism is needed to establish a new path to the destination by coordinating the untrusted links (i.e., the LC sets of nodes); and the LC set of a node has to be "reset" (set to empty) for either saving memory or when nodes lose state due to reboots. We address these issues in the next section and present an on-demand routing protocol based on LVC.

4.3 Link Vector Routing (LVR) Protocol

LVR uses route request (RREQ), route reply (RREP), and route error (RERR) messages similar to that of other on-demand routing protocols. LVR realizes the LVC condition in an on-demand context to maintain instantaneous loop-freedom. For this purpose, RREQs and RREPs carry the set of links that should not be trusted by any of the nodes originating or forwarding the message. The LC in a RREQ prevents any nodes from sending a RREP unless its stored path to the destination does not contain any link traversed by the RREQ or the LC

stated in the RREQ.

4.3.1 Information Stored and Exchanged

In the remainder of this paper, $X_D^A(t)$ denotes a value X that is stored at node A with respect to destination D at time t. The time is omitted when it refers to the current time. We use the superscripts req and rep to refer to RREQs and RREPs, e.g., X_D^{req} denotes a parameter X carried in a RREQ for destination D.

The routing-table entry for a destination D at a given node A specifies: (a) The successor to D (s_D^A), (b) the current link vector (LV_D^A), (c) the link coordination set (LC_D^A), (d) the lifetime of the entry (te_D^A), (e) the path cache ($pathcache_D^A$), and (f) a flag for link-vector cache use ($useLVforCache_D^A$). The path cache for a destination D consists of a set of path entries, each comprising of: (a) the link vector to an intermediate node I ($LV_D^{A\rightarrow I}$), (b) the estimated ttl to the destination ($estttl$), and (c) the expiry time for this path ($Lifetime_D^{A\rightarrow I}$). The path cache serves the purpose of collecting information from RREPs that are generated by intermediate nodes that have a loop-free path. The estimated ttl in the path cache helps set the RREQ expiry timer when a unicast source-routed RREQ is sent to coordinate the LC from the intermediate node to the destination.

Node A maintains a link cache ($lcache^A$), that contains tuples of link entries ($head$, $tail$, c, $expirytime$), with each entry consisting of the node addresses of the link head and tail entries, the associated cost, and the link life time. The $useLVforCache$ flag stored for a destination route-entry is used to determine whether the specific LV is valid and can be used to update the link cache.

Node A processing a RREQ for destination D identified by the unique source, request id pair $(src, rreqid)$ caches the aggregate LC set ALC_D^{req}, which we denote as $LC^A_{(src,rreqid)}$. This stored value is used to setup the route entry with the correct LC when the RREP is received for this RREQ.

4.3.2 Route Search and Maintenance

Sources establish routes to destination by flooding route request (RREQ) queries. Each RREQ for a destination (dst) carries the address of the originator (src) along with a unique identifier ($rreqid$) set by this originator so that the $(src, rreqid)$ is globally unique for destination dst. To determine which nodes can reply to a RREQ, each RREQ carries an aggregated link coordination set (ALC) consisting of the union of the LC's of the source and the nodes that relayed this RREQ. This ensures that generated RREPs can be used by the nodes to update their routing tables.

A node is said to be *active* for a computation (A, ID_A) for a destination if it is the source of a RREQ identified by $(src = A,\ rreqid = ID_A)$. Node A becomes active for a RREQ (A, ID_A) when it has data packets to send to destination D, but has no active valid route entry for D. A node is *engaged* in a RREQ computation (A, ID_A) by caching the corresponding previous hop address. The relaying node also caches the corresponding ALC from the RREQ; we denote the cached ALC at a node B for this RREQ as $LC^B_{(A,ID_A)}$. Otherwise, the node is *passive*.

At any given time, a node can be active in at most one RREQ for the same destination. The RREQ (A, ID_A) terminates when either node A receives a feasible reply satisfying LVC

for destination D or the timer for its RREQ expires. A node may be engaged in multiple RREQs for the same destination, but relays a RREQ (A, ID_A) only once by caching the identifier of the RREQ it forwards.

4.3.2.1 Finding loop-free paths using link coordination sets

The following two rules are used to manipulate the ALC of a RREQ at the source and relaying nodes:

Change Link vector Condition (CLVC): If node A has a successor $s_D^A = B$ for destination D and must change its successor because of a link failure, then it adds $LC_D^A \leftarrow LC_D^A \cup (A, B)$, and sets $LV_D^A \leftarrow \phi$. It issues a new RREQ carrying $ALC_D^{req} = LC_D^A$.

Relay Link Vector Condition (RLVC): Node A relays a RREQ for D if it has not previously processed the RREQ identified by the $(src, rreqid)$ pair, and sets $ALC_D^{req} \leftarrow ALC_D^{req} \cup LC_D^A$.

Carrying the untrusted link set LC in a RREQ, as in CLVC, allows the neighbor of a source to check if LVC can be satisfied at the source in a distributed fashion. However, RREQs usually span multiple hops; hence, as per RLVC, relay nodes must add their LC to the aggregated ALC in the RREQ. This allows a node generating a reply to check LVC in a distributed fashion along the entire path. For example, in Figure 4.2(c), if node B's RREQ is relayed by node X, then the RREQ's ALC is set to $\{$(BC),(XY)$\}$, which allows node Z to check LVC, locally, which will ensure that the generated reply will satisfy LVC at nodes X and B.

4.3.2.2 Minimizing floods using topology information

To limit the number of RREQs that need to be flooded, LVR uses the partial-topology information collected from link vectors to unicast RREQs to the destination, and uses the loop-free check of LVC to allow intermediate nodes to reply to RREQs. A RREQ contains a link vector LV used to direct the RREQ as follows:

- *Path Cache:* If LV is not empty, then LV is set to the best path (based on the metrics considered). Nodes relaying this RREQ must unicast it to the next-hop specified in LV. After the RREQ reaches a node having a valid active route to the destination, it is forwarded using the successor entries towards the destination.

- *Link Cache:* If a path can be determined using any standard path-selection algorithm (i.e., Dijkstra), then LV is set to that path. Nodes relaying this RREQ source-route the RREQ to the destination.

- *Expanding-ring flooding:* In this case LV is set to ϕ, and RREQ floods are issued with a controlled ttl. Nodes relaying the RREQ must append the traversed link to LV; for example, node B on receiving a RREQ from node A for destination D must set $LV_D^{req} \leftarrow LV_D^{req} \oplus (A, B)$.

If the path stated in the LV of a RREQ is incorrect or invalid, the RREQ is dropped at an intermediate node. To indicate that the RREQ must be unicasted, a flag bit 'S' is set in the RREQ. In all three cases, a corresponding *R*REQ expiry timer is set equal to 2 * per-hop-latency * hop-count. The value of hop-count is derived from LV, if it is valid. When a expanding ring-search is used, the hop-count is incremented suitably to flood an increased

portion of the network. If no RREPs are received for destination D after a number of attempts, a failure is reported to the upper layer.

4.3.3 Route Establishment

4.3.3.1 Generating RREPs

Route replies are generated in response to a previously unprocessed RREQ by either by the destination or an intermediate node that has a valid loop-free path. A RREP message contains the destination dst for which this RREQ was issued by a source, along with the unique identifier $(src, rreqid)$ for this RREQ. Each RREP contains LV and LC, which are set respectively to the link vector and the link coordination set for this destination at the node transmitting the RREP. A cost metric for the route may be implicitly derived from LV or carried separately in the RREP, i.e., hop-count. The lifetime of the route is also carried.

The destination replies for every unique RREQ flood once; however, intermediate nodes having a valid route to a destination can only reply if the following condition is satisfied. Intermediate nodes can issue two types of RREPs, which are distinguished using the coordinated 'Co' bit carried in the RREP flags.

Start Link vector Condition (SLVC): Node I can issue a RREP responding to a RREQ for destination D if I has an active route to D and $\nexists l \mid (l \in LV_D^I \wedge l \in ALC_D^{req})$. If $LC_D^I \subseteq ALC_D^{req}$, then node I sets $Co_D^{rep} = 1$.

SLVC serves as a distributed check of LVC for intermediate nodes to generate replies. The 'Co' bit is set if the second part of LVC is satisfied, i.e., the path is loop-free and the LCs of the nodes along the path are coordinated. The rationale behind using the 'Co' bit is to avoid

coordinating all paths to the destination as only one path will be setup for the route. Also, satisfying the first part of LVC means that the node has a loop-free path to the destination for this route search. Hence, the source collects intermediate replies which might be un-coordinated and saves them in the path cache. When the route fails later, the source can unicast a RREQ from the path cache and coordinate the path without requiring to flood the network. If the 'Co' bit is set, then the RREP also carries an additional parameter, the estimated route cost ($estcost$), which is the hop-count to the destination from the node issuing the RREP. The $estcost$ helps a source set up a correct RREQ expiry timer when unicasting a RREQ from the path cache to an intermediate node for coordination.

4.3.3.2 Processing RREPs

RREPs that have the coordinated bit set (Co=1) can be used by the nodes to update their routing table entry if LVC is satisfied. Node A on receiving a RREP identified by (S, ID_S) with Co=1 from node B for destination D sets the following: $s_D^A \leftarrow B$; $LV_D^A \leftarrow (A, B) \oplus LV_D^{rep}$; $LC_D^A \leftarrow LC_D^A \cup LC_{(S,ID_S)}^A$.

RREPs received without the 'Co' bit set are added to the path cache. A node A receiving a RREP for destination D, adds the link vector LV_D^{rep} to $pathcache_D^A$ and sets the $esttl$ in the cache for this entry to $estcost$ + cost(LV_D^{rep}).

To illustrate an example using Figure 4.2(c): Assume node X has an active route to D with LV_D^X={XY,YD}. If link BC fails, node B sends a RREQ to node X which applies SLVC to see that it can generate a RREP with Co bit not set because LC_D^X={}. The RREP will carry a $estcost$ of 2 (for XY,YD). Node B will add this RREP to its path cache, and then coordinate

the path XYD to setup the LCs. On the other hand, if the LCs were already coordinated, the Co bit can be set, and node B can use the RREP directly to update its route entry.

4.3.4 Handling Failures

When the link to next hop B for destination D is no longer used, then node A adds the successor link entry to its link coordination set, $LC_D^A \leftarrow LC_D^A \cup (A, B)$. Routing table maintenance and route error message handling are much the same as discussed in Chapter 3 for FLR.

4.3.5 Resetting Link-Coordination Sets

LVR requires a safe reset mechanism for the LCs maintained at the nodes as they are critical to detecting outdated link vectors and avoiding the formation of loops. The LC at a node for a destination needs to be reset when (i) state is lost due to a reboot or if the routing table was purged to save memory, (ii) a RREQ packet to be generated or relayed will exceed packet size due to the size of the ALC, or (iii) a RREQ is relayed with a ALC that out-dates the currently stored LV. If reliable communication among neighbors is possible, then techniques proposed for clearing old link-state information [39] can be used. However, adhoc networks usually operate on top of an unreliable link layer; hence, we use our framework based on destination sequence numbers that we introduced in Chapter 2.

We give a brief description of how per-destination sequence numbers are used in LVR as a 'reset' mechanism. Each node stores a per-destination sequence number, and the associated LV and LC. The base LVR protocol operation for a destination can be considered as one running

on the same per-destination sequence number at all the nodes, and represents a pseudo time-frame. Whenever a node requires to reset its LC, it requests a destination sequence number reset, as specified in our framework, by forcing the destination to answer the request. The node then acquires a increased sequence number for the destination, effectively advancing its pseudo time-frame. It is intuitive that nodes only learn newer sequence numbers, advancing their pseudo time-frames. Hence, loss of information about links that should not be trusted does not affect loop-freedom as all upstream nodes are invalidated from new routing computations when a reset is acquired by a node. This is equivalent to maintaining different timed-instances of untrusted links (LC) and trusted links (LV) at the nodes for a destination; however, links can be trusted again when they have a fresher time-stamp.

4.3.6 Cache Maintenance

The lifetime of links in the cache, which is used to calculate paths for source-routing RREQs, are refreshed by a pre-defined parameter $active\_time\_period$ on two events: (i) a RREQ or RREP that was processed carried the link in the traversed link vector, and (ii) a data packet was forwarded using a route entry that has this link in its stored link vector, and $useLVforcache$ is *true*.

When a RERR carrying a failed link is received, $useLVforcache$ is set to *false* for all route entries with stored link vectors using this link, and the cache lifetime is set to expired for this link. Also, when an source-routed RREQ fails, then the associated links in the link vector are invalidated.

Note that the links added to the cache in LVR are always fresh as they only carry

the path traversed by the RREP. Even RREPs from intermediate nodes carry only the link-vector from the source to the intermediate node. This is in contrast to DSR, where the RREPs can carry stale paths which are created by appending old outdated paths stored at intermediate nodes towards the destination.

As an optimization, sequence numbers associated with the head node of each link can be used to determine the freshness of the updates. Because LCs are essentially the set of links that have failed, the cache can be kept more accurate if LCs along with sequence numbers for a destination are transmitted in the RERRs.

4.4 Example

To illustrate the operation of LVR, we assume that all link costs are unity, and hence are not included in the description. For brevity, the use of sequence numbers is addressed only for LC resets.

4.4.1 LVR operation

4.4.1.1 Setting up routes

Fig. 4.3 (a) shows the directed acyclic successor graph for destination D in a six-node network operating LVR as the routing protocol. Nodes A and E have an active flow to D and the corresponding link vectors (LV), and the link coordination set (LC) are as shown at time t. This route set up results from a RREQ flood. We assume the link coordination set LC is empty at all nodes. The generated RREP is forwarded along the reverse path, where LVC is satisfied

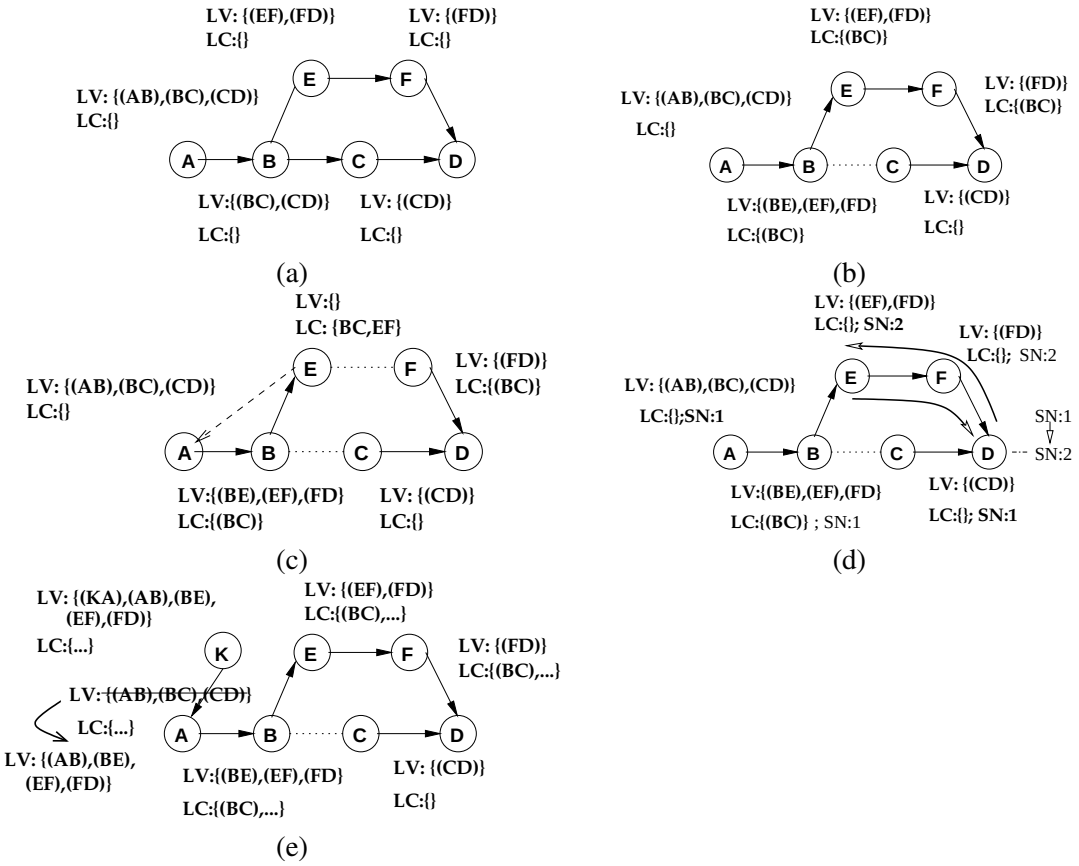


Figure 4.3: LVR protocol operation

at all the nodes.

4.4.1.2 Re-establishing routes

We now illustrate how LVR coordinates the link vectors at nodes along the new successor path after link BC fails at time $t_1 > t$. If not performing a local repair, node B sends a RERR to A that may or may not be received. For the purposes of this discussion, unless stated otherwise, we assume that RERRs are never delivered. Node B adds $LC_D^B \leftarrow LC_D^B \cup (B, C)$ to keep track of the failed links for destination D. Node B attempts to re-establish the route

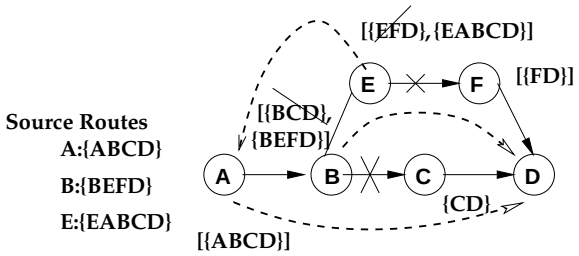


Figure 4.4: Loops using DSR packet salvaging operation

by sending a RREQ with $ALC_D^{req} = \{(B, C)\}$. Node E can reply to the RREQ when it receives the RREQ because SLVC is satisfied: $\nexists l \mid l \in ALC_D^{req} = (B, C) \wedge l \in LV_D^E = \{(E, F), (F, D)\}$, but the successor path $\{E, F, D\}$ has not been coordinated: $ALC_D^{req} \not\subseteq LC_D^E$. Hence, as an intermediate node, node E issues a RREP with $Co = 0$, $estcost = 2$, and $LV_D^{rep} = \{(B, E)\}$.

Node B cannot use the RREP it receives for updating its routing tables because $Co_D^{rep} = 0$, so it adds it to the pathcache $pathcache_D^B$ with $estttl = 3$ (from estcost + cost(LV_D^{rep})). To coordinate the successor path from B to D, node B unicasts the RREQ (B, ID_B) to E with $ALC_D^{req} \leftarrow \{(B, C)\}$, $S = 1\}$. When nodes E and F process the RREQ, because the S-bit is set, they will unicast the RREQ along to F and D, respectively; both nodes cache for (B, ID_B), $LC \leftarrow \{(B, C)\}$. This RREQ essentially serves as a unicast "probe" message, and because it causes no network-wide flood, it saves bandwidth.

The destination D generates a RREP with $Co_D^{rep} = 1$ when it receives the request. When the RREP traverses the reverse path, nodes E and F can accept the RREP because it satisfies LVC. Following update rules, node F will set $LC_D^F \leftarrow LC_D^F \cup LC_{(B,ID_B)}^A = \{(B, C)\}$; and, similarly, node E does the same. When node E relays the RREP with $LC_D^{rep} = \{(B, C)\}$,

it satisfies LVC at B, and as shown in Figure 4.3(b), the successor path $\{B, E, F\}$ to destination D is setup with the correct LC coordination.

4.4.1.3 Preventing loops and counting-to-infinity

When link EF fails at time $t_2 > t_1$. Assume that the link from node E to A was temporarily down and comes up. Node E queries a RREQ with $ALC_D^{req} = \{(B, C)\}$. However, node A cannot answer the RREQ because SLVC is not satisfied, i.e., $(B, C) \in ALC_D^{req}$ $\wedge (B, C) \in LV_D^A$. This avoids loops that could have been formed because of the stale link vector possessed by A. Subsequently, even though the network is partitioned, no nodes can learn new routes from upstream nodes, and hence no counting-to-infinity can occur. The network state is shown in Figure 4.3(c).

4.4.1.4 Reseting link coordination sets

Figure 4.3(d) shows the state after a reset along path EF. Assume that at time t_1, all the nodes had learned a destination sequence number of 1 for D. Now, say node E requires to reset its LC for one of the reasons listed in Section 4.3.5. Node E sends a RREQ requesting a destination sequence number reset, and the RREP (for simplicity) is generated by D after incrementing its sequence number to 2 and traverses the reverse path $\{E, F\}$. Nodes E, and F update their respective sequence numbers to 2, and can set their LC to . Note that despite losing their LC, no loops can form because the set of upstream nodes $\{A, B\}$ have a sequence number of one and their replies are no longer believed by E and F. The sequence numbers help prevent count-to-infinity or loops when LC information is lost.

4.4.2 DSR packet salvaging

The same sequence of events discussed in Section 4.4.1 can lead to loops, if DSR with packet salvaging is used as the routing protocol. Figure 4.4 shows the directed acyclic successor graph for destination D, with node A and B having active flows. The source routes cached at each of the nodes for destination D is as shown within *[]* and the data packets are source-routed using the path $\{A, B, C, D\}$ at A. With the subsequent link breaks of BC and EF, nodes B and E salvage the packets along their source routes $\{E, F, D\}$ and $\{E, A, B, C, D\}$, respectively. The local packet salvaging mechanism ends up looping the packets along $\{A, B, E, A\}$ as shown in figure.

4.4.3 Use of topology information in LVR

4.4.3.1 Cache freshness

In the previous DSR example, note that node E can learn and use the stale route $\{A, B, C, D\}$ even if its not salvaging a packet. This problem has been analyzed [40] previously, where the authors proposed time-stamping links using epochs of sequence numbers to determine freshness; however, it still cannot avoid stale link information being learned the first time. On the other hand, LVR implicitly maintains cache freshness because of the untrusted links carried in the RREQs and the coordination mechanism. For example, node E's RREQ will carry $ALC = \{(BC)\}$ which ensures that SLVC will not be satisfied at node A to generate a RREP. This prevents RREPs with stale routes from being issued.

To illustrate how only fresh correct links are added to the cache, we refer to Figure 4.3

(e), where node B initially was using the path $\{BCD\}$ before it switched to $\{EFD\}$ and node A has not been notified of this link failure because the RERR was lost. Let node K query A with a RREQ carrying $ALC = \{...\}$ (the actual values are not significant for this example). Assume SLVC is satisfied, but the path is un-coordinated, so the RREP has Co-bit unset. The link cache is only updated with the link (K, A), as intermediate RREPs only carry the path traversed. Note that this in contrast to DSR where the stale path $\{ABCD\}$ will be appended in the RREP. To use that path, in LVR, node K will send a unicast RREQ to A for co-ordination along the path. This RREQ will be forwarded along the path $\{ABEFD\}$, and when the RREP is received, only this traversed path is added to the link cache. As an indirect implication of the coordination mechanism, only fresh links are added to the cache. This is despite the fact that the intermediate node A had a stale route at the time it initiated a RREP. The cache may later become stale due to mobility, but nodes never learn stale routes directly from RREPs.

4.4.3.2 Minimizing floods

Using the same network configuration in Figure 4.3 (e), assume that the active routes to destination D are along paths $\{ABCD\}$ and $\{EFD\}$. If node K sends a RREQ, it will receive RREPs with Co-bit unset from nodes B and C, assuming that SLVC is satisfied. Node K will store paths to B and C in its path cache for D. Then, node K will co-ordinate the best path, say EFD. If link KE fails at a later time, node K will co-ordinate the path BCD by looking up the path cache. If this route fails, node K checks for any paths that can be computed using the link cache. In essence, node K makes use of all available topology information to set up loop-free paths before resorting to flooding RREQs to the destination.

4.5 Analysis

For the purposes of this analysis, we use the sequence number framework discussed in Section 4.3.5. Our proofs consider only the case when the destination sequence numbers are equal; it is easy to see that fresh higher sequence numbers over-ride the link-vector conditions so that older outdated sequence numbers cannot be used.

Theorem 19. *In LVR, along any successor path $P = \{n_k, ..., n_1\}$ that exists to the destination at any point of time, it is true that $LC_{n_1}^{n_i} \subseteq LC_{n_1}^{n_{i-1}}$, for $i \in \{2, k\}$.*

Proof. For path P to exist at a given time t, it must be true that that all nodes in P have a valid successor. According to LVC, node n_i can make n_{i-1} it successor for destination n_1 only if it accepts a RREP rep with $LC_{n_1}^{n_{i-1}} \subseteq LC_{n_1}^{n_i}$. Hence if path P were to exist at time t, then it must be true that every node n_i ($i \in \{2, k\}$) must have accepted a RREP from n_{i-1} with it should have satisfied $LC_{n_1}^{n_{i-1}} \subseteq LC_{n_1}^{n_i}$. □

Theorem 20. *Routing tables are loop-free in LVR at any instant.*

Proof. Let node I initiate a RREP for destination D and let the RREP traverse the path P=$\{n_1,...,n_j\}$, where $n_1 = I$ (maybe the destination D) and $n_j = A$. Let the path from I to D be Q=$\{m_1, ..., m_k\}$, where m_1= D and $m_k = I$ and Q can be null if $n_1 = D$.

For a loop to form, node A must be on the path Q and must change successors after processing the RREP generated by I. Because RREQs and RREPs travel loop-free paths, path P must be loop free. Therefore, node A cannot form a loop after processing I's RREP if $n_1 = D$. If $n_1 \neq D$ we must show that it is impossible for A to be on I's successor graph.

Assume that path Q exists at time t_0 and is loop free and node A changes successors after processing I's RREP at time t_1. Node I sends its RREP along the loop-free path P to A. Let A be some node m_i, $1 < i < k$. For A to accept the RREP received from n_2, it must be true that $LC_D^A \subseteq LC_D^{rep} \subseteq LC_D^I$, and $Co_D^{rep} = 1$. To initiate a RREP with $Co_D^{rep} = 1$, it must be true at node I that $\nexists l, l \in ALC_D^{req} \wedge l \in LV_D^I$ when the RREP was issued. By the same argument as in Theorem 18 and using the invariant condition of Theorem 19, it must be true that $\exists l, l \in LC_D^A \wedge l \in LV_D^I$. Hence, node A cannot accept the RREP, and no loops can be formed in this case. □

Theorem 21. *In a connected component G, partitioned from destination D, all nodes must invalidate their routing table entries for D, in the presence of link failures and node reboots.*

Proof. The proof follows directly by the use of the sequence number framework [14], [15] when state loss is incurred by nodes. On the other hand, if no state is lost, then default RERR propagation should invalidate routes because nodes never re-learn routes from upstream nodes as shown in Theorem 20 and there can be no cyclic propagation of RERRs.

□

Theorem 22. *In a error-free stable connected network, LVR ensures that a node starting a route computation (A, ID_A) for a destination D establishes a successor path to the destination in finite time.*

Proof. This proof is similar to that used for previous protocols [8] (Theorem 5, pp. 61) [14] (Theorem 5). The RREQ (A, ID_A) elicits a reply with 'Co' set (from the destination, can also be from a source-routed RREQ that was sent), and all nodes along the path will update their LC

from their cached tuple for (A, ID_A). This ensures that all nodes along the reverse path accept the RREP. □

4.6 Performance

We present results for LVR over varying loads and mobility. The protocols used for comparison are four on demand protocols DSR, AODV, AODV-PA, and FLR, which reflect the state of the art in on-demand routing, and OLSR which is a pro-active link state protocol. Simulations are run in Qualnet[44]. The parameters are set as in [41]. No optimizations were used for LVR, link cache expiry times were set to 2 seconds, and the maximum size of link coordination set in control packets allowed before requiring a reset was set at 20.

4.6.1 Simulation Setup

Simulations are performed on two scenarios, (i) a 50-node network with terrain dimensions of 1500m x 300m, and (ii) a 100-node network with terrain dimensions of 2200m x 600m. Traffic loads are CBR sources with a data packet size of 512 bytes. Load is varied by using 10 flows (at 4 packets per second) and 30 flows (at 4 packets per second). The MAC layer used is 802.11 with a transmission range of 275m and throughput 2 Mbps. The simulation is run for 900 seconds. Node velocity is set between 1 m/s and 20 m/s. We use two sets of traffic characteristics: (i) random flows have a mean length of 100 seconds, distributed exponentially, and (ii) fixed flows that last the entire simulation time. For the fixed flows, we only show results for 30-flows because we did not notice any shift in trend from the one observed for the 10-flows

scenario with exponential flow distribution. Each combination (number of nodes, traffic flows, scenario, routing protocol and pause time) is repeated for nine trials using different random seeds.

4.6.2 Performance Metrics

We address four performance metrics. *Delivery ratio* is the ratio of the packets delivered per client/server CBR flow. *Latency* is the end to end delay measured for the data packets reaching the server from the client. The *network load* is the total number of control packets divided by the number of received data packets. *Data hops* is the number of hops traversed by each data packet (including initiating and forwarding) divided by the total number of received packets in the network.

4.6.3 Performance Discussion

Tables 4.1 and 4.2 summarize the results of the different metrics by averaging over all pause times for the 50 and 100 node networks with random exponentially distributed flows. Table 4.3 summarizes the same set of metrics for 50 and 100-node networks with fixed flow distributions. The columns show the mean value and 95% confidence interval. All our performance discussions focus on the average case because the confidence intervals overlap atleast slightly in most cases. The packet delivery ratio, the end-to-end delay, and the control overhead over various pause times for 100-node networks with 30-flows is shown for random exponential flows in Figure 4.5, and for fixed flows in Figure 4.6. The vertical bars in the graphs indicate the 95% confidence intervals.

Table 4.1: Performance average over all pause times for 50 nodes network for 10-flows and 30-flows (random)

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
LVR	10	0.995±0.001	0.024±0.004	0.342±0.079	2.528±0.175
FLR	10	0.994±0.002	0.019±0.003	0.256±0.060	2.500±0.171
AODV	10	0.994±0.002	0.016±0.003	0.270±0.066	2.576±0.179
AODV-PA	10	0.994±0.002	0.017±0.005	0.268±0.065	2.583±0.190
DSR	10	0.940±0.027	0.041±0.047	0.220±0.095	2.677±0.185
OLSR	10	0.887±0.040	0.012±0.001	1.937±0.220	2.456±0.175
LVR	30	0.855±0.036	0.482±0.171	1.569±0.429	2.739±0.232
FLR	30	0.820±0.051	0.491±0.180	2.005±0.722	2.717±0.268
AODV	30	0.765±0.055	1.010±0.356	4.423±1.289	2.951±0.324
AODV-PA	30	0.754±0.055	1.092±0.351	4.343±1.191	3.002±0.336
DSR	30	0.683±0.059	4.760±1.073	0.410±0.140	3.625±0.308
OLSR	30	0.798±0.034	0.883±0.311	0.713±0.069	2.478±0.161

The summarized results are indicative of LVR's comprehensive performance over all mobility pause times with both flow distributions. In the 10-flow scenarios, LVR and AODV have statistically equivalent packet delivery ratios. LVR has a slightly higher control-overhead and latency because nodes with no sequence number information about the destination have to force RREQs to be answered by the destination [15]. However, both DSR and OLSR exhibit poor packet delivery, which indicates that corrupted topology information due to mobility affects the accuracy of the routes in both the protocols even at very light-loads. The problem with DSR lies in the use of source-routes to deliver packets. Source-routes are invalidated due to topology changes and data packets are forced to be dropped by intermediate nodes. This problem is aggravated by the use of an optimization to learn source-routes carried in data packets that can be stale. OLSR's performance suffers from the temporary loops that can exist until the latest topology change is disseminated to all nodes in the network. The loops cause unwanted transmission of data packets, thereby congesting the physical medium which in-turn affects the

Table 4.2: Performance average over all pause times for 100 nodes network for 10-flows and 30-flows (random)

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
LVR	10	0.988±0.004	0.099±0.109	1.326±0.352	3.712±0.301
FLR	10	0.989±0.004	0.043±0.007	0.838±0.251	3.671±0.340
AODV	10	0.988±0.004	0.036±0.009	0.897±0.236	3.744±0.293
AODV-PA	10	0.988±0.004	0.033±0.006	0.896±0.237	3.802±0.310
DSR	10	0.876±0.050	0.099±0.057	0.859±0.353	4.257±0.317
OLSR	10	0.821±0.063	0.022±0.002	11.795±1.575	3.583±0.256
LVR	30	0.724±0.035	0.906±0.160	6.557±1.078	4.204±0.311
FLR	30	0.648±0.047	0.874±0.188	8.347±1.795	4.355±0.370
AODV	30	0.608±0.051	1.455±0.385	18.298±13.069	4.751±0.434
AODV-PA	30	0.582±0.052	1.682±0.445	16.888±7.394	4.918±0.417
DSR	30	0.618±0.049	5.125±0.782	1.243±0.405	6.141±0.499
OLSR	30	0.612±0.041	3.371±0.532	5.423±0.669	4.014±0.277

dissemination of topology updates.

The results in the 30-flow scenarios show much more disparity in the average performance of the protocols although most results lie in overlapping confidence intervals. The packet delivery graphs across various mobility pause times (4.6(a), and 4.5(a)) show that LVR has a very consistent performance across all pause times, even close to that of OLSR at very low mobility pause time considering that OLSR performs very well in low-mobility scenarios. The same trend is reflected in the latency characteristics of LVR, where it is even better than the pro-active OLSR. LVR has the highest average packet delivery, and the lowest latency in the 30-flow scenarios. LVR delivers 11% more data packets than AODV, DSR, and OLSR in both the 30-flows scenarios with 100-nodes. LVR also convincingly outperforms FLR. The control overhead of LVR is smaller than that of AODV and AODV-PA. Although OLSR and DSR have smaller control overhead than LVR and AODV, they cannot be directly compared. The reason is that OLSR's control overhead is independent of the traffic flow characteristics, and DSR floods

fewer RREQs because of the optimization through which it can learn source-routes carried in data packets. A higher control overhead usually results in congesting the medium, which in turn causes delay when transmitting data packets. The low-delay characteristics of LVR can be attributed to its control overhead, which is smaller than the overhead incurred by AODV. In the case of OLSR, although it incurs small control overhead, it renders higher delays than LVR because of the looping of data packets. The higher delays in DSR are due to the buffering of data packets while new source-routes are being found.

Table 4.3: Performance average over all pause times for 50-nodes and 100-nodes network with 30-flows (fixed)

Protocol	Size	Delivery Ratio	Latency (sec)	Net Load	Data Hops
LVR	50	0.833±0.059	0.424±0.237	1.504±0.544	2.837±0.295
FLR	50	0.758±0.089	0.584±0.342	2.858±1.313	2.903±0.386
AODV	50	0.738±0.083	0.866±0.473	5.044±1.874	3.084±0.424
AODV-PA	50	0.727±0.082	0.963±0.505	4.942±1.739	3.149±0.441
DSR	50	0.581±0.081	3.422±0.813	0.430±0.156	3.809±0.406
OLSR	50	0.772±0.042	0.842±0.413	0.727±0.072	2.534±0.187
LVR	100	0.738±0.068	0.612±0.217	5.295±1.639	4.187±0.461
FLR	100	0.623±0.100	0.735±0.314	9.307±3.622	4.504±0.707
AODV	100	0.608±0.088	1.060±0.402	16.812±8.743	4.731±0.696
AODV-PA	100	0.572±0.085	1.322±0.480	16.679±6.446	4.957±0.695
DSR	100	0.476±0.099	3.865±1.150	1.208±0.453	6.177±0.733
OLSR	100	0.585±0.066	2.761±1.062	5.541±0.811	4.074±0.452

We were unable to obtain tight confidence intervals for the control overhead of AODV in the summarized results. We believe that the reason for the excessive control overhead and poor packet delivery (Figs. 4.6 and 4.5) in the scenarios with very high mobility are due to flooding of RREQs that have to reach the destination to be answered, which is due to the way in which AODV handles sequence numbers after route failures. This congests the network which in-turn aggravates the problem because more routes get broken triggering new RREQ floods.

AODV with path accumulation (AODV-PA) performs about the same as AODV. This is due to the fact that the optimization allows route entries to be setup for relay nodes, and this will only improve performance when there are flows towards the relay nodes within the small window of time before the routes expire. These results are in direct correlation with the performance results shown in [16], where it was found to show improvements only in huge networks with many flows and was proposed as an optimization in such a case. LVR collects the same path as in the case of AODV-PA; however, it uses the topology information for its routing decisions in-contrast to AODV-PA's technique of learning routes to relay nodes and discarding the path information.

The superior performance attained with LVR compared to the other on-demand routing protocols is due to the reduction of control overhead, which directly results in lower contention for the medium and better packet delivery with lower latency. Tables 4.4 and 4.5 show the average success rate of the source-routed RREQs (S-RREQ) and coordination achieved by sending unicast probes (I-RREQ) to an intermediate node from the path cache for all pause times in networks of 100-nodes with random and fixed-flow traffic distributions, respectively. The results show that, on the average, ˜50% of source-routed RREQs and unicast probes for coordination are successful in establishing routes. Also shown are the total number of acceptable RREPs received by sources, and the fraction of RREPs that are generated by source-routed RREQs (S-RREP) and coordination through intermediate nodes (I-RREP). On an average, S-RREPs and I-RREPs comprise about 30-35% of total replies received in the random-flows scenario, and about 40-45% in the fixed-flows scenario. Note that both S-RREPs and I-RREPs are generated without requiring a network-wide flood to the destination. This explains LVR's

low control-overhead and short latency, because most routes are established by using path-information or by coordinating with an intermediate node across a relatively small number of hops. The fixed-flow scenarios seem to benefit more from the cached path information, because the set of destinations are fixed throughout the simulation, which means that more intermediate nodes have information about the destination. Although FLR uses topology information for routing decisions, it does not attempt to maintain the correctness of the paths, and hence optimizations such as the source-routing from link caches do not render much benefit (as shown by results in Chapter 3). By determining the correctness of the link vectors, LVR is able to use the cache more effectively.

The *data hops* metric provides a measure of the accuracy of the routes used for forwarding. The data hops metric reflects the number of hops traversed by each data packet whether or not it is delivered. By correlating the packet delivery ratio with the data hop count, a notion of how many packets were actually delivered to the destination can be gauged. All protocols have statistically equivalent data hops across all scenarios, which means that in protocols delivering less packets, some of the packets are dropped at the intermediate nodes. In AODV, packets are dropped at intermediate nodes due to falsely triggered route failures or temporary loops with heavy congestion. Data packets in OLSR can temporarily traverse loops before being delivered or can get dropped due to lack of routes. Stale source routes in DSR, due to mobility, cause packets to be dropped at the intermediate nodes. The data hops of LVR indicate that packets are forwarded more accurately to the destination.

Table 4.4: LVR - RREP statistics and success rate of RREQs in 100-nodes network with 30 fixed-flows

	Success Rate (%)		Fraction of RREPs (%)		
Pause Time	S-RREQ	I-RREQ	S-RREP	I-RREP	Total RREPs
0	45.96	41.87	20.67	21.57	11676.44
50	53.57	51.82	19.74	20.73	9504.50
100	52.91	47.93	18.83	23.16	9408.00
200	51.53	47.78	19.51	22.75	9587.22
300	54.17	48.24	17.85	24.75	10345.56
500	50.77	45.78	18.56	25.73	11662.56
700	56.80	51.46	17.36	26.92	10878.00
900	60.88	55.67	14.12	29.24	10405.11

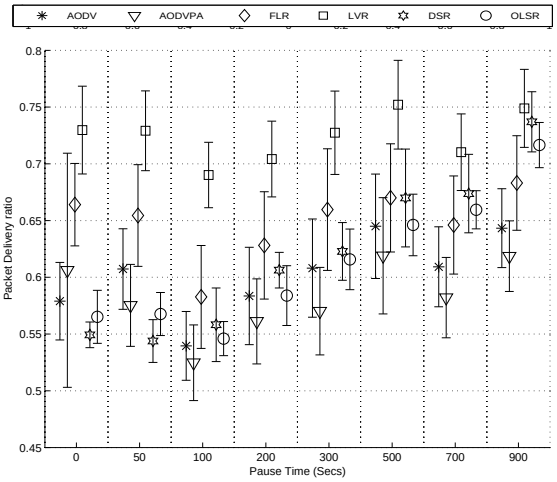
Table 4.5: LVR - RREP statistics and success rate of RREQs in 100-nodes network with 30 random-flows

	Success Rate (%)		Fraction of RREPs (%)		
Pause Time	S-RREQ	I-RREQ	S-RREP	I-RREP	Total RREPs
0	44.61	41.93	13.08	21.95	11367.44
50	44.45	41.13	13.30	22.40	11819.44
100	42.29	37.66	13.03	24.17	12883.44
300	46.01	39.91	13.21	24.43	12218.00
300	49.09	44.59	12.31	24.85	11513.78
500	52.30	45.19	11.36	24.94	10049.78
700	44.48	39.62	9.77	22.09	11609.67
900	54.28	46.82	10.71	25.75	10151.00

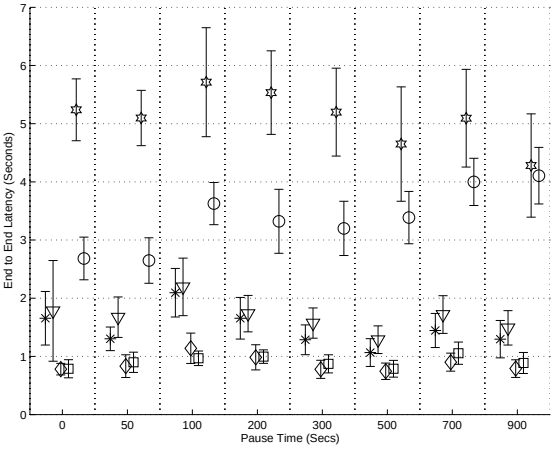
4.7 Conclusion

We presented a new sufficient condition to achieve loop-freedom in topology-based routing protocols by maintaining a set of links that can no longer be trusted when nodes reroute through new paths. Applying this condition, we have presented a new on-demand routing protocol for ad hoc networks, the Link Vector Routing (LVR) protocol, which collects link vectors in its control messages similar to other proposals (AODV-PA, DSR, OLIVE, and FLR); however, LVR maintains instantaneous loop-freedom using the collected topology information regarding the links that form part of available paths to destinations, as well as links that should not be used

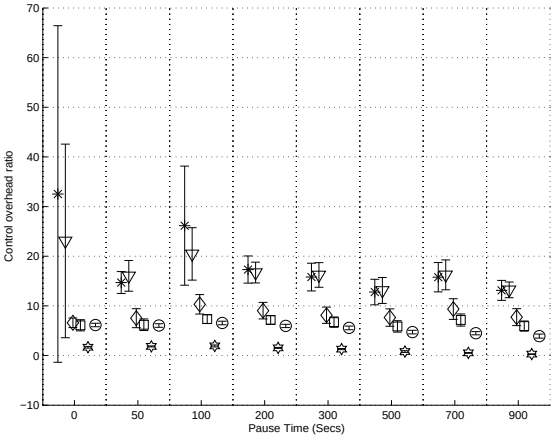
in establishing paths to destinations. LVR does not require source routes for data packets, reliable message exchanges among neighbors, or data-packet filtering. Extensive simulations are used to illustrate that LVR achieves better performance than AODV, AODV-PA, DSR, FLR, and OLSR. Analyzing simulation results for LVR show that a sizable percentage of RREPs received by sources are from RREQs that they source-routed or coordinated through an intermediate node, which can be attributed to the better topology information collection and maintainence, avoiding network-wide RREQ floods to reach the destination.



(a) Packet Delivery

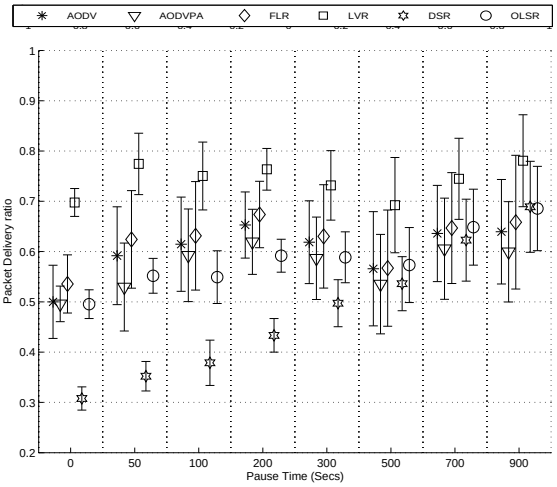


(b) Latency

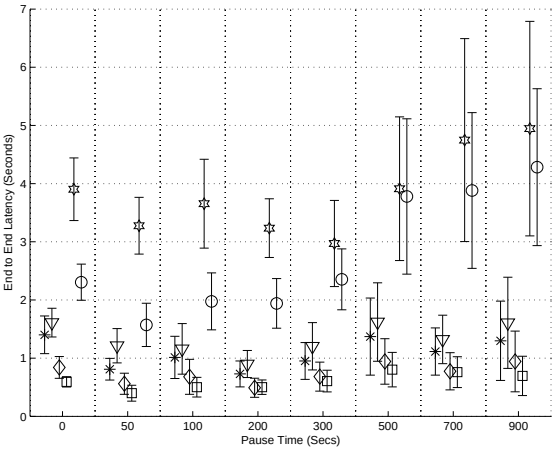


(c) Control Overhead

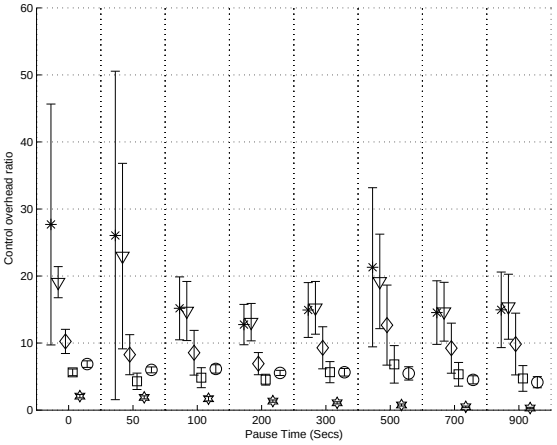
Figure 4.5: Random (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

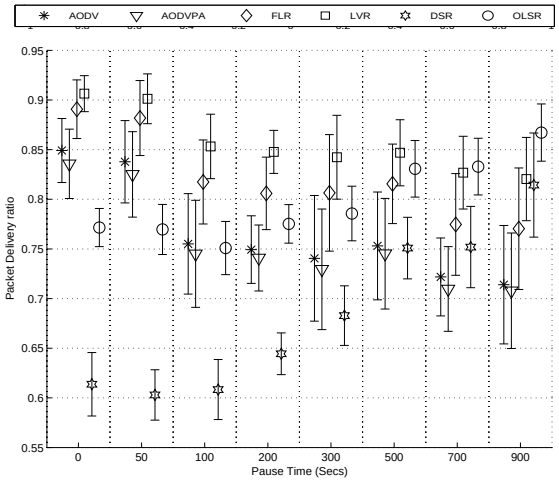


(b) Latency

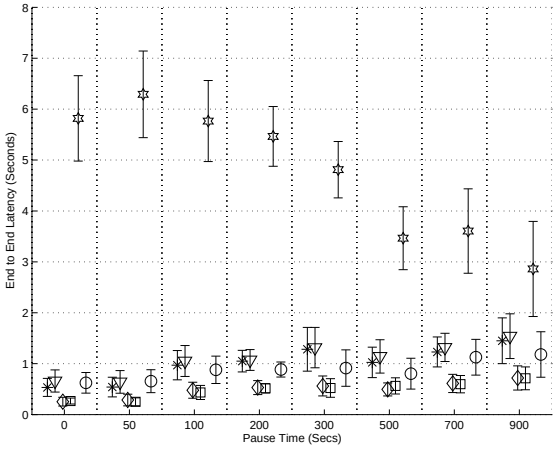


(c) Control Overhead

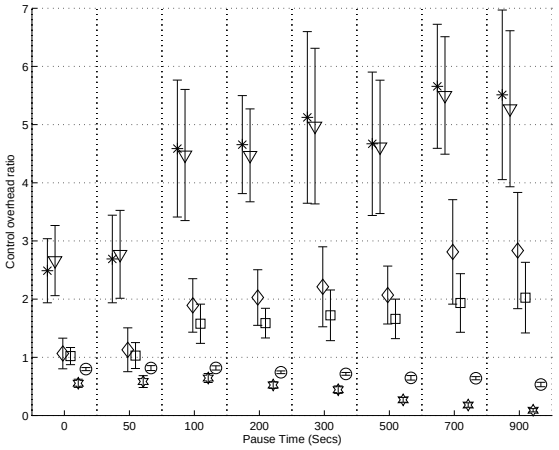
Figure 4.6: Fixed (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

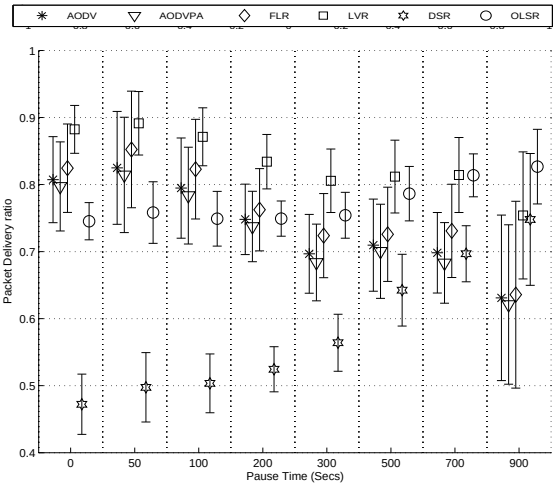


(b) Latency

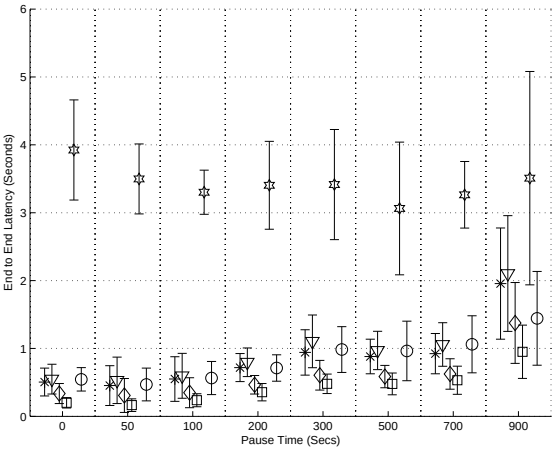


(c) Control Overhead

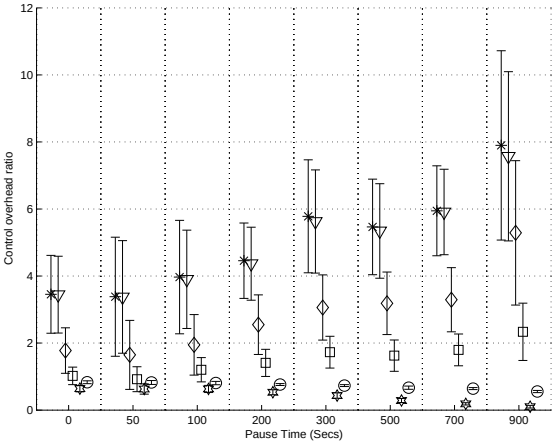
Figure 4.7: Random (50-nodes, 30-flows, 120 pps)



(a) Packet Delivery



(b) Latency



(c) Control Overhead

Figure 4.8: Fixed (50-nodes, 30-flows, 120 pps)

Chapter 5

Routing using source sequence numbers

In any on-demand routing protocol, sources flood route requests to build routes to destinations, and each new route request is identified uniquely with a source-sequenced label (SSL) consisting of the source identifier and a locally generated sequence number. As a route request propagates, it creates a directed acyclic graph, because nodes relay each route request only once. In this chapter, we present the first framework for loop-free on-demand routing in ad hoc networks that is based directly on SSLs, rather than on independent mechanisms, which has been the way in which prior on-demand routing protocols have been designed. Our framework supports hop-by-hop packet forwarding, maintains routing tables that are always loop-free, and operates correctly in the presence of state loss, node failures, and unreliable message delivery.

Section 5.1 presents an approach for loop-free routing in which the destination must answer all route requests (RREQ). Each node relaying a RREQ associates a *relay-sequenced label* (RSL) with the SSL of the RREQ it forwards. The route replies (RREP) traversing the reverse paths along the DAG built by the RREQs cause nodes to switch successors and activate

the route along this path. However, nodes can be associated with multiple RREQs, and the directed graph associated with the aggregate of all such RREQs need not be acyclic. Hence, to ensure loop-free routes to a destination, nodes use the RSLs to accept a RREP for only one SSL and drop the other replies, which essentially activates successor entries for a destination within a single DAG created by a RREQ SSL. This constitutes the first component of our framework.

Section 5.2 introduces the concept of a *viable successor set* (VSS), which is the set of nodes that a given node can safely pick as loop-free successors for a given destination. We use this concept to extend the approach introduced in Section 5.1 to allow intermediate nodes to send replies to RREQs without creating loops. The basis for this approach is the use of label sets consisting of un-ordered collections of SSLs and RSLs with which nodes can identify viable successors towards destinations.

Section 5.3 presents an alternative approach to extending the basic approach of Section 5.1. This approach uses the SSL and distance to a destination as a new label for the destination, and uses it together with RSLs to ensure loop-free routing while allowing nodes with valid routes to destinations to answer RREQs.

Section 5.4 shows how instantiations of our new framework perform compared to two on-demand protocols (AODV, DSR) and a proactive link state protocol (OLSR). Results are presented for 50 and 100-node networks with random traffic flows. Section 5.5 provides our concluding remarks.

5.1 On-Demand Routing Using Source Sequence Numbers and Destination Replies

We present an approach for on-demand routing based on the SSLs and RSLs carried in RREQs, and on RREPs issued only by destinations. We call this approach DSLR (destination-controlled, source-sequenced labeled routing).

5.1.1 Information Stored

Node A has a unique address for itself (A) and maintains a 64-bit source sequence number ID_A that is strictly increasing, even after reboots. This can be achieved by deriving the source sequence number from a real-time hardware clock. Node A also stores a boot-strap identifier, BID_A, which is set to the initial value of ID_A when the node starts-up.

At node A, the routing-table entry for destination D consists, at a minimum, of the successor (next-hop) s_D^A and the start-identifier SID_D^A, which is used for checking if a certain neighbor can be used as a loop-free successor. SID_D^A can be set to either of these two parameters depending on the loop-free condition being used: (a) $ID_D^{\star A}$, which is the last known value of $ID_A(t)$ at time t when node A last added or updated its routing entry for D, or (b) CID_D^A, which is the last known $ID_A(t)$ that node A assigned to a route computation for D and modified its routing table for D using this computation. Nodes can use optional metrics for choosing successors. Possible optional entries include the route-cost (d_D^A), life-time of the route, and state of the route entry (rt_D^A) (which can be valid or invalid). The route entry for a destination can be purged at any time-instant to save memory. When there is no routing-state,

the value of SID_D^A is set to $ID_A(t)$, where t is the time that node A first originates or relays a RREQ for D; otherwise, it is considered to be invalid (∞).

5.1.2 Control Signaling

The basic signaling of DSLR consists of *route requests* (RREQ) sent by sources and forwarded by relays, and *route replies* (RREP) sent only by the destinations. Nodes notify route failures and broken links using *route errors* (RERR). Each RREQ is identified with a *source-sequenced label* (SSL) that consists of a *(source, identifier)* pair, where *source* is the node address originating the RREQ, and *identifier* is the source sequence number ID created by that node. RREPs generated by destinations in response to RREQs must carry the SSL used to identify the RREQ. In addition to the SSL, each RREQ and RREP must carry the *relay-sequenced label* (RSL), which is locally assigned by the node relaying the RREQ or RREP, and is only significant within the one-hop neighborhood.

Node A is said to be *active* in a route computation for destination D (i.e., the RREQ) when it initiates a RREQ that is uniquely identified by the pair (A, ID_A). A node relaying a RREQ (A, ID_A) originated by another node is said to be *engaged* in the RREQ. A node that is not active or engaged in a route computation for destination D is said to be *passive* for that destination. At any given time, a node can be the origin of at most one RREQ for the same destination. The RREQ (A, ID_A) terminates when either node A attains a viable successor for destination D or the timer for its RREQ expires.

We define the following five rules for nodes to search for routes to destinations. RERR handling is omitted for brevity and is identical to ones used in previous on-demand routing

protocols [19]. We use the notation ID_A to denote the current source sequence number at node A, and id_A^{req} and id_A^{rep} to denote the value carried in a RREQ or RREP.

- *Rule 1:* Node A must increment its ID_A each time it relays or originates a RREQ.
- *Rule 2:* If node A requires a route to destination D, it issues a RREQ identified by SSL (A, ID_A). At the originating node, the RSL and the SSL for the same destination are identical.
- *Rule 3:* If node B ($\neq D$) receives a RREQ identified by SSL (A, id_A^{req}) from neighbor C, it caches the RSL (C, id_C^{req}) for this SSL. Node B processes a RREQ identified by a SSL only once, and forwards a RREQ to its neighbors with the SSL created by the source of the request and its own RSL (B, ID_B).
- *Rule 4:* When node D receives a RREQ from neighbor I that was issued by source A for node D itself, it sends a RREP carrying the same SSL (A, id_A^{req}) of the RREQ, and the RSL (I, id_I^{req}).
- *Rule 5:* When node I receives a RREP for destination D identified by SSL (A, id_A^{rep}) and carrying a RSL (I, id_I^{rep}), it must drop the reply if $id_I^{rep} < BID_I$. Otherwise, the RREP can be used, if it is feasible, to update its routing table. Then, if the RREP can be relayed, node I must find the pair (B, id_B^{req}) it cached for this (A, id_A^{rep}), and send a RREP to neighbor B with RSL (B, id_B^{req}) and SSL (A, id_A^{rep}).

Theorem 23. *If rules 1 to 5 are followed, RREQs and RREPs do not loop in an error-free network.*

Proof. For a given route computation (A, ID_A), a node may be passive, engaged, or active. A node can become active in a route computation at most once, because it maintains the identifiers it assigns to the RREQs it originates. A router can engage in a route computation only when the corresponding RREQ identified by (A, ID_A) has not been previously processed. Hence, any RREQ can traverse only a directed acyclic graph (DAG), which may be a directed tree if no node relays the RREQ more than once, and any path traversed by a RREQ is free of loops.

Because RREPs are forwarded along the reverse path traversed by the corresponding RREQs, it follows that the RREPs must traverse loop-free paths. □

Note that RREQs may loop if nodes lose their cached state for a computation. However, RREPs will not loop if nodes adhere to Rule 5 when processing them as shown by this theorem.

Theorem 24. *Under any condition, Rule 5 ensures thats RREPs do not loop.*

Proof. For a RREP identified by SSL (I, ID_I) to loop, it must be true that the cached reverse hop entries stored at nodes for SSL (I, ID_I) in a path P must form a cycle C. However, because of Theorem 23, this is not possible unless one or more nodes on path P lost cached state and reprocessed the RREQ (I, ID_I). Assume that a node $n \in P$ associated a RSL (n, ID_n) with SSL (I, ID_i) at time t_0, and loses state and restarts operation at time $t > t_0$. It is true that $BID_n(t) > ID_n$. Node n on reprocessing the RREQ after time t must have associated a new RSL (n, ID'_n) with the RREQ (I, ID_I), such that $ID'_n \geq BID_n(t)$. To show that RREPs do not loop, we have to show that cycle C created by the cached reverse path entries of the nodes for SSL (I, ID_I) does not exist at time t. According to Rule 5, node n can only process RREPs

carrying a RSL (n, id_n), such that $id_n \geq BID_n(t)$. Hence, at time t, node n will not process a RREP carrying the first processed RSL (n, ID_n), and does not participate in the cycle created by the reverse route entries for SSL (I, ID_I). Therefore, RREPs can never loop. □

5.1.3 Sufficient Conditions for Loop Freedom

Theorem 23 shows that the RREPs travel loop-free reverse paths because the RREQ identified by a unique SSL build a tree rooted at the source. It is easy to see that no loops can form if every node in that reverse path to the source is engaged in only one route computation for the destination. Based on this observation, we obtain two sufficient conditions for loop-free routing when multiple RREQs are present: The *Source Sequence-number Condition* (SSC), allows nodes to only accept a single computation within a window of computations. After accepting the computation, the node drops all other route computations inside that window, and processes only computations from a new window. The *Source Sequence-number Ordered Condition* is a tighter version of the SSC condition that allows sequentially ordered computations within the same window to be accepted, which in-turn renders more RREPs usable.

We use the following terminology: $id(A)^A_{DB}$ denotes the id from RSL (A, id) in the RREP for destination D sent by node B to node A. $id(B)^A_D$ is the id from RSL (B, id) in the RREP that A receives from or transmits to a neighbor.

5.1.4 Source Sequence-number condition

Source Sequence-number condition (SSC): Node A can change its current successor for destination D to node B at time t, if $id(A)^A_{DB}(t) \geq SID^A_D(t)$, where $SID^A_D(t) = ID^{\star A}_D(t)$.

Source sequence-number Relay Condition (SRC): Node A must relay a RREP received from neighbor B to neighbor C that is engaged in this SSL RREQ computation for D, iff node A processed the received RREP and updated its routing table entry for destination D.

We establish the following Lemmas (2 and 3) when nodes obey rules 1 to 5, and use SSC for switching successors.

Lemma 2. $SID_D^A(t_1) \leq SID_D^A(t_2)$, *where* $t_1 < t_2$.

Proof. We know that $SID_D^A(t_1) = ID_D^{\star A}(t_1)$. If node A never updates its routing table till time t_2, then $SID_D^A(t_2) = SID_D^A(t_1)$. On the other hand if node A updates route-entry for D at time t_2, then it can only be the case that $ID_D^A(t_2) > ID_D^A(t_1)$. Therefore $SID_D^A(t_2) = ID_D^{\star A}(t_2) > ID_D^{\star A}(t_1)$. Even if there is a reboot or state loss after time t_1, it is still true at time $t_2 \geq t > t_1$, that $SID_D^A(t) = ID_A(t) > ID_D^{\star A}(t_1)$. Hence, the lemma is true. □

Lemma 3. *The event of reporting the value of* $id(B)_D^A$ *at time* t *to neighbor* B *has a causal relation (denoted by* $\rightsquigarrow$*) with the event that node* A *uses a value of* $id(A)_D^A$ *to update its routing table at time* t^-*, where* $t = t^- + \epsilon$*, assuming that* ϵ *is the processing time for updating the route table.*

Proof. By Rule 5, node A can report a RREP at time t only if it used a RREP to update its routing table. Hence, node A must have used the value of $id(A)_D^A$ reported by a RREP at time t^-. By Rule 3, node A must have a stored value of node B's RSL for this RREP identified by an unique SSL. So, by Rule 5, node A reports the value of $id(B)_D^A$ at time t from the cache after updating its routing table for a time ϵ. Therefore, the events are causally related. □

Theorem 25. *If nodes use SSC to change successors, no routing table loops can form.*

Proof. The proof is by contradiction. Assume that, before time t, the directed successor graph for destination D, which we denote by $S_D(G)$ is loop-free at every instant, and a loop $L_D(G)$ is formed at time t. It is easy to see that no loops can be formed unless atleast one node changes its successor at time t to a node that is upstream of itself in $S_D(G)$.

Assume that $L_D(t)$ is formed when node i makes node a its new successor $s_D^i(t)$ after processing an input event at time t, where $b = s_D^i(t_b) \neq a$ and $t_b < t$. Now, $P_{aD}(t)$ must include $P_{ai}(t)$.

Let $P_{ai}(t)$ consist of the chain of nodes {a= s[1,new], s[2,new], ..., s[k,new],...,i} as shown in Figure 5.1. The notation indicates that node s[k, new] is the k^{th} hop in the path $P_{ai}(t)$ at time t, and has node s[k+1,new] as its successor at time t.

The last time that node s[k,new] updates its routing table entry up to time t and sets $s_D^{s[k,new]}$=$s[k+1, new]$ is denoted by $t_{s[k+1,new]}$, where $t_{s[k+1,new]} \leq t$. Therefore, it is true that $s_D^{s[k,new]}(t_{s[k+1,new]})$= $s_D^{s[k,new]}(t)$.

Because nodes joining P_{aD} do not switch to any new successors afterwords, it is also true that

$$SID_D^{s[k,new]}(t_{s[k+1,new]}) = SID_D^{s[k,new]}(t)$$

The time when node s[k,new] sends a reply that constitutes the last reply from such a node that is processed by node s[k-1, new] up-to time t is denoted by $t_{s[k+1,old]}$. Node s[k, new]'s successor at time $t_{s[k+1,old]}$ is denoted by s[k+1, old]. Note that $t_{s[k+1,old]} \leq t_{s[k+1,new]} \leq t$, and that $s[k+1, old]$ need not be the same as $s[k+1, new]$. It is also true that $SID_D^{s[k,new]}(t_{s[k+1,old]}) \leq SID_D^{s[k,new]}(t_{s[k+1,new]})$.

From Rule 5 and Lemma 3, when a node $s[k, new]$ relays a RREP to node $s[k -$

$1, new]$ at time $t_{s[k+1,old]}$ after updating its routing table at time $t^{-}_{s[k+1,old]}$, it must be true that

$$id(s[k,new])_{D}^{s[k,new]}(t^{-}_{s[k+1,old]}) < ID_{D}^{\star s[k,new]}(t_{s[k+1,old]})$$

Because SSC must be satisfied when node s[k,new] $\in P_{aD}(t)$ makes node s[k+1,new] $\in P_{aD}(t)$ its successor at time $t_{s[k+1,new]}$, it must be true that

$$id(s[k,new])_{Ds[k+1,new]}^{s[k,new]}(t) = id(s[k,new])_{Ds[k+1,new]}^{s[k,new]}(t_{s[k+1,new]})$$
$$\geq SID_{D}^{s[k,new]}(t_{s[k+1,new]})$$

For a loop to be formed after t, it must be true that $P_{aD}(t)$ exists. We now derive the following inequality along the path $P_{ai} \subset P_{aD}$ at time t, if nodes satisfy SSC when switching successors. We use Lemmas 2 and 3.

$$SID^i_D(t) = ID^{\star i}_D(t) \leq id(i)^i_{Da}(t) = id(i)^a_D(t_{s[2,old]}) \rightsquigarrow id(a)^a_D(t^-_{s[2,old]}) <$$

$$ID^{\star a}_D(t_{s[2,old]}) \leq ID^{\star a}_D(t_{s[2,new]}) = SID^a_D(t_{s[2,new]}) \leq id(a)^a_{Ds[2,new]}(t) =$$

$$id(a)^{s[2,new]}_D(t_{s[3,old]}) \rightsquigarrow ... \rightsquigarrow id(s[k,new])^{s[k,new]}_D(t^-_{s[k+1,old]}) <$$

$$ID^{\star s[k,new]}_D(t_{s[k+1,old]}) \leq ID^{\star s[k,new]}_D(t_{s[k+1,new]}) =$$

$$SID^{s[k,new]}_D(t_{s[k+1,new]}) \leq id(s[k,new])^{s[k+1,new]}_{Ds[k,new]}(t) =$$

$$id(s[k,new])^{s[k+1,new]}_D(t_{s[k+1,old]})$$

$$\rightsquigarrow ... \rightsquigarrow id(i)^i_D(t^-_b) < ID^{\star i}_D(t_b) \leq ID^{\star i}_D(t) = SID^i_D(t)$$

The invariant conditions along this path lead to the erroneous conclusion that $SID^i_D(t) < SID^i_D(t)$. Hence, no loops can be formed when SSC is applied. □

5.1.5 Source Sequence-number Ordered condition

Source Sequence-number Ordered condition (SSOC): Node A can change its current successor for destination D to node B at time t, if $id(A)^A_{DB}(t) \geq SID^A_D(t)$, where $SID^A_D(t) = CID^A_D(t)$.

The value of CID^A_D is updated as follows: When node A switches successors for destination D using SSOC at time $t^+ = t + \epsilon$, where ϵ is route table processing time, the value of $CID^A_D(t^+)$ is set to $id(A)^A_{DB}(t)$.

Source sequence-number Ordered Relay Condition (SORC): Node A must relay a RREP received from neighbor B to neighbor C that is engaged in this SSL RREQ computation

for D, iff $id(A)^A_D(t) \leq CID^A_D(t)$, where $id(A)^A_D(t)$ is the id from the RSL (A, id) in the RREP being processed.

To prove loop-freedom, we first establish the following Lemmas (4 and 5) when nodes obey rules 1 to 5, and use SSOC for switching successors and SORC for relaying RREPs.

Lemma 4. *$SID^A_D(t_1) \leq SID^A_D(t_2)$, where $t_1 < t_2$.*

Proof. We know that $SID^A_D(t_1) = CID^A_D(t_1)$. If node A never updates its routing table till time t_2, then $SID^A_D(t_2) = SID^A_D(t_1)$. On the other hand if node A updates route-entry for D at time t_2 using SSOC, then it can only be the case that $CID^A_D(t_2) > CID^A_D(t_1)$. Even if there is a reboot or state loss after time t_1, it is still true at time $t_2 \geq t > t_1$, that $SID^A_D(t) = ID_A(t) > CID^A_D(t_1)$. Hence, the lemma is true. □

Lemma 5. *The event of reporting the value of $id(B)^A_D$ at time t to neighbor B has a causal relation (denoted by $\rightsquigarrow$) with the event that node A uses a value of $id(A)^A_D$ to update its routing table at time t^-, where $t = t^- + \epsilon$, assuming that ϵ is the processing time for updating the route table.*

Proof. By Rule 5, node A can report a RREP at time t only if it used a RREP to update its routing table. Hence, node A must have used the value of $id(A)^A_D$ reported by a RREP at time t^-. By Rule 3, node A must have a stored value of node B's RSL for this RREP identified by an unique SSL. So, by Rule 5, node A reports the value of $id(B)^A_D$ at time t from the cache after updating its routing table for a time ϵ. Therefore, the events are causally related. □

Theorem 26. *If nodes use SSOC to change successors, no routing table loops can form.*

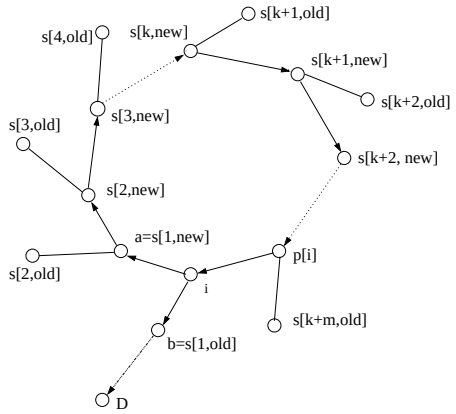


Figure 5.1: Loop in G

Proof. The proof is by contradiction. Assume that, before time t, the directed successor graph for destination D, which we denote by $S_D(G)$ is loop-free at every instant, and a loop $L_D(G)$ is formed at time t. It is easy to see that no loops can be formed unless atleast one node changes its successor at time t to a node that is upstream of itself in $S_D(G)$.

Assume that $L_D(t)$ is formed when node i makes node a its new successor $s_D^i(t)$ after processing an input event at time t, where $b = s_D^i(t_b) \neq a$ and $t_b < t$. Now, $P_{aD}(t)$ must include $P_{ai}(t)$.

Let $P_{ai}(t)$ consist of the chain of nodes {a= s[1,new], s[2,new], ..., s[k,new],...,i} as shown in Figure 4.1. The notation indicates that node s[k, new] is the k^{th} hop in the path $P_{ai}(t)$ at time t, and has node s[k+1,new] as its successor at time t.

The last time that node s[k,new] updates its routing table entry up to time t and sets $s_D^{s[k,new]}$=$s[k+1, new]$ is denoted by $t_{s[k+1,new]}$, where $t_{s[k+1,new]} \leq t$. Therefore, it is true that $s_D^{s[k,new]}(t_{s[k+1,new]})$= $s_D^{s[k,new]}(t)$.

Because nodes joining P_{aD} do not switch to any new successors afterwords, it is also true that

$$SID_D^{s[k,new]}(t_{s[k+1,new]}) = SID_D^{s[k,new]}(t)$$

The time when node s[k,new] sends a reply that constitutes the last reply from such a node that is processed by node s[k-1, new] up-to time t is denoted by $t_{s[k+1,old]}$. Although, note that node s[k-1,new] may relay replies sent by node s[k,new] from time $t_{s[k+1,old]}$ to t.

Node s[k, new]'s successor at time $t_{s[k+1,old]}$ is denoted by s[k+1, old]. Note that we only know that $t_{s[k+1,old]} \leq t$ and $t_{s[k+1,new]} \leq t$, and that $s[k+1, old]$ need not be the same as $s[k+1, new]$. It is also true that $SID_D^{s[k,new]}(t_{s[k+1,old]}) \leq SID_D^{s[k,new]}(t_{s[k+1,new]})$.

From SORC and Lemma 5, when a node $s[k, new]$ relays a RREP to node $s[k-1, new]$ at time $t_{s[k+1,old]}$, it must be true that

$$id(s[k, new])_D^{s[k,new]}(t_{s[k+1,old]}) \leq CID_D^{s[k,new]}(t_{s[k+1,old]})$$

Because SSOC must be satisfied when node s[k,new] $\in P_{aD}(t)$ makes node s[k+1,new] $\in P_{aD}(t)$ its successor at time $t_{s[k+1,new]}$, it must be true that

$$\begin{aligned} id(s[k, new])_{Ds[k+1,new]}^{s[k,new]}(t) &= id(s[k, new])_{Ds[k+1,new]}^{s[k,new]}(t_{s[k+1,new]}) \\ &\geq SID_D^{s[k,new]}(t_{s[k+1,new]}) \end{aligned}$$

For a loop to be formed after t, it must be true that $P_{aD}(t)$ exists. We now derive the following inequality along the path $P_{ai} \subset P_{aD}$ at time t, if nodes satisfy SSOC when switching successors. We use Lemmas 4 and 5.

$$SID^i_D(t) = CID^i_D(t) \leq id(i)^i_{Da}(t) = id(i)^a_D(t_{s[2,old]}) \rightsquigarrow id(a)^a_D(t_{s[2,old]}) \leq$$

$$CID^a_D(t_{s[2,old]}) = SID^a_D(t_{s[2,old]}) \leq SID^a_D(t_{s[2,new]}) \leq id(a)^a_{Ds[2,new]}(t) =$$

$$id(a)^{s[2,new]}_D(t_{s[3,old]}) \rightsquigarrow ... \rightsquigarrow id(s[k,new])^{s[k,new]}_D(t_{s[k+1,old]}) <$$

$$CID^{s[k,new]}_D(t_{s[k+1,old]}) = SID^{s[k,new]}_D(t_{s[k+1,old]}) \leq$$

$$SID^{s[k,new]}_D(t_{s[k+1,new]}) \leq id(s[k,new])^{s[k+1,new}_{Ds[k,new]}(t) =$$

$$id(s[k,new])^{s[k+1,new]}_D(t_{s[k+1,old]})$$

$$\rightsquigarrow ... \rightsquigarrow id(i)^i_D(t_b) \leq CID^i_D(t_b) = SID^i_D(t_b) \leq SID^i_D(t)$$

The invariant conditions along this path lead to the erroneous conclusion that $SID^i_D(t) < SID^i_D(t)$. Hence, no loops can be formed when SSOC is applied. □

5.1.6 Basic Route Maintenance

5.1.6.1 Route Establishment

Routes to destinations are established on demand when data packets destined for that destination are received. Node A that is *active* for destination D buffers such data packets. However, if node A is *passive* for destination D then it must become *active* and issue a RREQ as per Rule 2. Node A maintains a *RREQ timer* that is set to (*2.ttl.latency*) for every destination for which is it active, where ttl is the time-to-live of the broadcast flood and latency is the

estimated per-hop latency of the network. If no usable RREPs are received, node A resends new RREQs with an increased ttl after the expiry of its timer. If node A does not receive a RREP for destination D after a number of attempts, a failure is reported to the upper layer. The number of hops that a RREQ can traverse is controlled externally from the RREQ by means of the TTL field of the IP packet in which a RREQ is encapsulated, or by other means.

5.1.6.2 Updating Routing Tables

Node I sets $s_D^I \leftarrow B$ when it accepts a RREP from neighbor B that satisfies SSC or SSOC. If it has an associated route metric, it updates $d_D^A \leftarrow d_D^{rep} + lc_B^A$. Note that nodes can chose to accept only RREPs that will result in shorter-cost paths, although it is not necessary.

5.1.7 Termination Properties

5.1.7.1 Source Sequence-number condition

Theorem 27. *All nodes in a connected component G not containing destination D invalidate their route entries for node D within a finite time.*

Proof. From Theorem 23, RREPs generated by the destination cannot traverse loops. A finite time t after node D is partitioned from nodes $n \in G$, all RREPs must have been processed at the sources that originated the RREQs, and no more RREPs can be present in the network. The DASG for D defined by the successor entries of nodes in G is loop-free (Theorem 25), and in a finite time all nodes in the DASG must be notified with a route error (RERR) stating the un-reachability of D. □

Theorem 28. *In a stable error-free connected network, a source S will establish a route to a destination D in finite time.*

Proof. From Theorem 23, RREPs generated by the destination D will travel a loop-free reverse back path to the source S. If every node along the path updates its routing table for D and relays the RREP, then the theorem is true. However, if a node is *engaged* in multiple route computations for D, then it may not satisfy SSC, and the RREP will be dropped. However, in such a case, if a node is engaged for 'n' different route computations, atleast one of them succeeds. Because along any path to the destination from the different sources, one RREP is always relayed, atleast one source must always establish a path to the destination irrespective of how many other route computations are on-going at the same time. Given that there are only a finite number of sources (nodes) in the network, and they retry new RREQs upon failure, eventually all sources must establish routes to the destination. □

5.1.7.2 Source Sequence-number Ordered condition

Theorem 29. *In a connected component G, not containing the destination D, all nodes will invalidate their route entries for node D within a finite time.*

Proof. From Theorem 23, RREPs generated by the destination cannot travel in loops. A finite time t after node D is partitioned from nodes $n \in G$, and all RREPs have been processed at the sources that originited the RREQs, no more RREPs will be present in the network. The DASG for D defined by the successor entries of nodes in G is loop-free (Theorem 26), and by default RERR propogation, all nodes in the DASG will be notified with a RERR stating the unreachability of D. □

Theorem 30. *In a stable error-free connected network, a source will establish a route to a destination in finite time.*

Proof. From Theorem 23, RREPs generated by the destination D will travel a loop-free reverse back to the source S. For a RREP not to traverse the reverse path, one of the nodes along the path must not relay it because SORC is not satisfied. Inorder for that to be true at a node n_i, it must receive a RREP with RSL (n_i, id) such that $id > CID_D^{n_i}$. However, such a case will satisfy SSOC, and node n_i must use this RREP to update its routing table, and, in-turn SORC will automatically be satisfied. This shows that the RREPs will either cause nodes to update their routing tables and relay them or they will be relayed because SORC is satisfied. Therefore, the RREPs traversing the reverse path will always reach the source, and all nodes along the path will have a valid route to the destination. □

5.1.8 Non-caching Option

Caching of the RSL associated with a RREQs SSL can be avoided at the relay nodes if the RREQs and RREPs themselves carry the entire list of RSLs generated at every relay node. This is beneficial when nodes have limited storage capacity. The message structure of such a RREQ or RREP would be a tuple $\{SSL, RSL_1, RSL_2, ..., RSL_n\}$, where SSL is assigned by the originating node, and each RSL_i, where $i \in \{1, ..., n\}$, is appended by the relaying node n_i and is carried as a list, rather than being cached at the intermediate nodes along the path. It is also possible for mixed-mode operation where some set of nodes can operate with the cache, while the ones that cannot can force the previous hop RSL to be carried in the message. For example, node n_i can flag a bit indicating that $RSL_{n_{i-1}}$, along with its new RSL_{n_i} must be

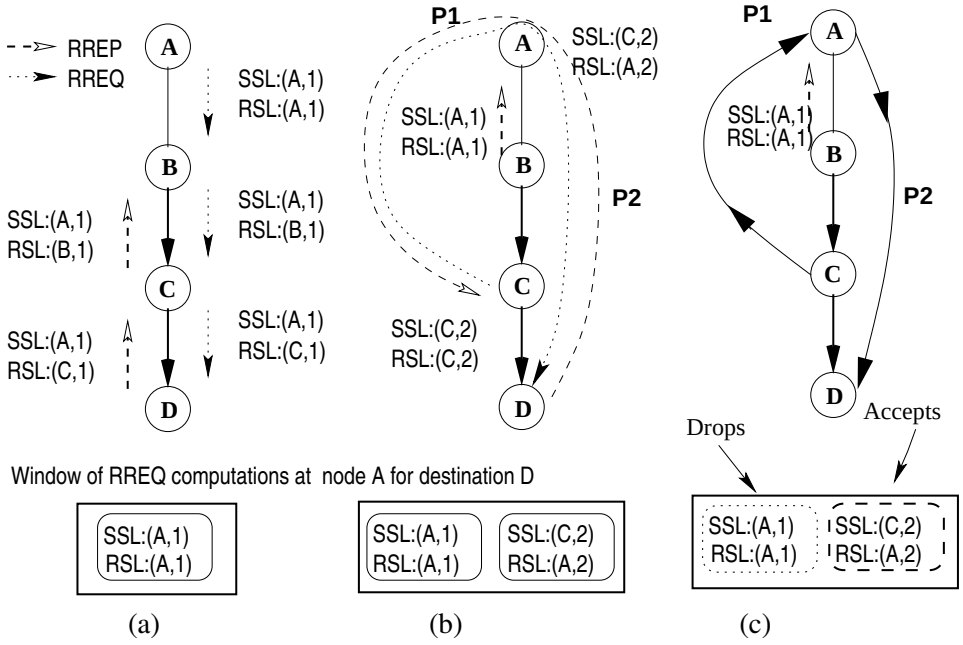


Figure 5.2: Using SSLs and RSLs with SSC

carried in the RREQ. This ensures that enough information is present in the RREP generated so that node n_i can process and relay the RREP to node n_{i-1}.

5.1.9 Example with SSC

Figure 5.2(a) shows the sequence of events in a four-node network when node A initiates a RREQ for destination D identified by SSL (A,1) along with a RSL (A,1) (that is the same as the SSL at the origin). Node B caches the RSL (A,1) for the SSL (A,1) and sends the RREQ with its RSL (B,1). Similarly, node C caches the RSL (B,1) and sends out a RREQ with RSL (C,1). Destination D generates a RREP which is processed by nodes B and C. Assuming that the network has just begun operation, when the nodes A, B, and C transmitted the RREQ they must set themselves a value of one for SID_D^A, SID_D^B, and SID_D^C, respectively. Node

C can accept the RREP (A,1) with a RSL (C,1) because SSC is satisfied and will switch its successor to node D. Similarly, node B will switch successors to C. Nodes B and C will set their respective SID_D^B and SID_D^C to two after updating their routing tables for D. Now, assume that the RREP relayed by node B is queued at the MAC layer at time t_1. Figure 5.2(b) shows the sequence of events after time t_1, when node C can no longer reach D due to a link failure. Node C sends a new RREQ that traverses the paths $P1$ and $P2$ through node A to reach destination D. The RREQ is identified by SSL (C,2) and node A relays it with a RSL (A,2). Note that node A still has a stored value of one for SID_D^A.

Figure 5.2(c) shows the state of the network at a time $t_2 > t_1$ when a RREP is issued by the destination identified by $(C, 2)$ and the nodes along path $P1$ and $P2$ update their routing tables to establish a route to node D. Node A sets $SID_D^A \leftarrow 3$ to establish its successor path along $P2$ after updating its routing table. When the RREP identified by SSL (A,1) queued at node B is finally transmitted and received by node A, it can form a loop if node A decides to switch successors to B. However, because SSC must be satisfied, node A can only accept RREPs carrying a RSL starting from (A,3) as $SID_D^A = 3$. Therefore, the RREP will be dropped and no loops are formed.

Figure 5.2 also shows the window of RREQ computations that node A is engaged or active for destination D. Node A becomes a part of two different DAGs created by the RREQs. However, the directed graph formed by the aggregate of RREQs with SSL (A,1) and (C,2) is not acyclic. Inside this computation window, node A becomes only a part of the DAG created by RREQ SSL (C,2), and drops the SSL (A,1). The window can consist of more RREQ computations, but only one of them is used, and the rest of the computations in the current

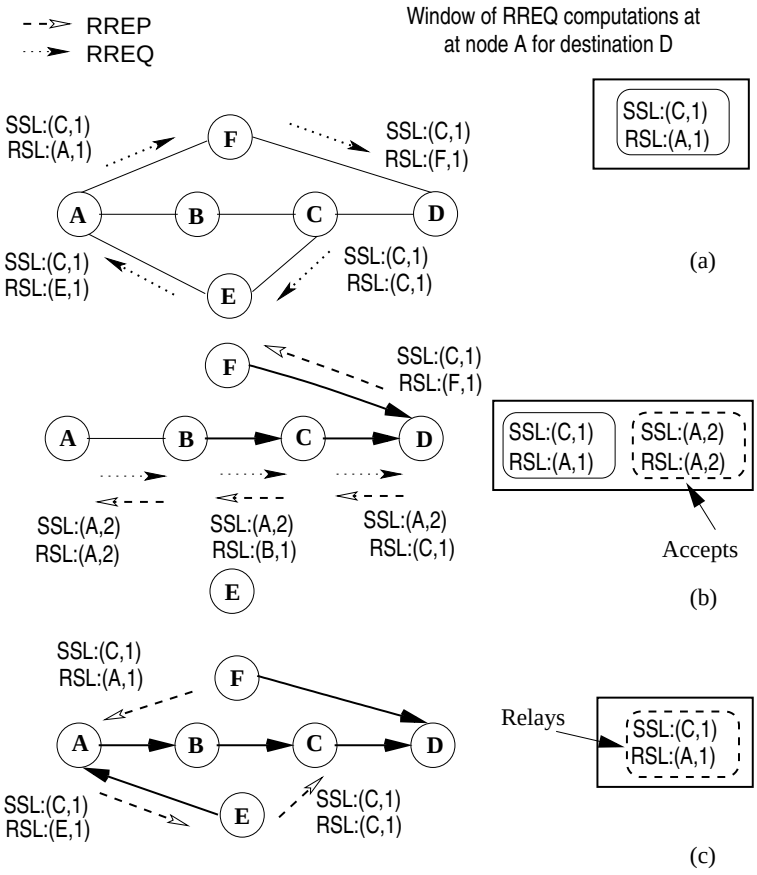


Figure 5.3: Using SSLs and RSLs with SSOC

window are dropped.

5.1.10 Example using SSOC

Figure 5.3(a) shows the sequence of events in a six-node network when node C initiates a RREQ for node D. We assume that node D does not receive the RREQ broadcast from C due to a transmission error. The RREQ identified by SSL (C,1) is forwarded by nodes E and F to node D with their respective RSLs (E,1) and (F,1).

On receiving the RREQ, destination D generates a RREP identified by SSL (C,1) with

a RSL (F,1). Node F can accept the RREP because SSOC will be satisfied. Assume that at the same time, node A requires a route for destination D and sends a RREQ identified by SSL (A,2) that traverses the path BCD. Similar to previous RREQ traversals, nodes B and C forward with their respective RSLs (B,1) and (C,2) which are cached at the receiving nodes. Figure 5.3(b) shows these sequence of events, and also the network state when the RREP generated for this RREQ identified by SSL (A,2) establishes the routing path along ABCD because SSOC is satisfied at each of the nodes.

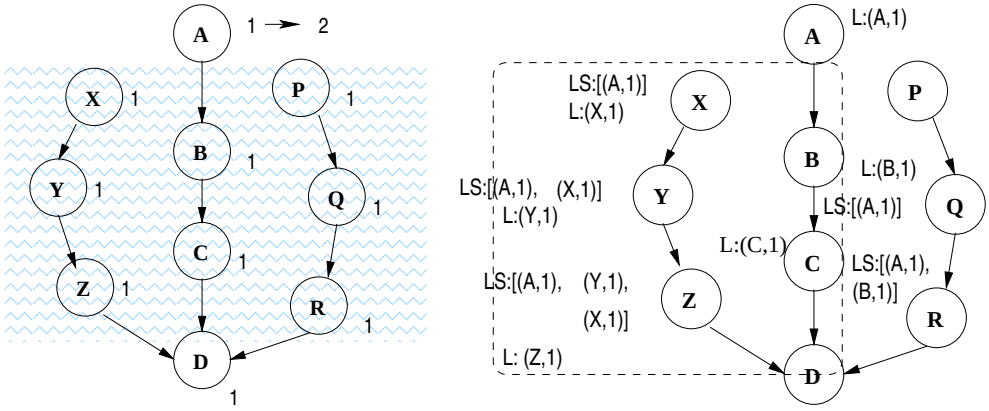
Figure 5.3(c) shows the events when node F transmits the RREP for the RREQ computation $(C, 1)$. When node A receives the computation, it cannot accept it to update its routing table, but as per the relay condition (SORC), but still relays it to node E which accepts the RREP and sets up its routing path to D. Note that, if SSC is used, the RREP would have been dropped at node A. This prevents nodes from establishing routes on every route computation that they are engaged or active for. After the RREQ timer expiry, node E or any other node has to subsequently attempt another RREQ flood and establish a route along path $ABCD$ or AFD.

5.2 On-Demand Routing Using Source Sequence Numbers and Replies from Nodes with Valid Routes

We extend the basic framework of the previous section by deriving labels out of the SSL and RSLs carried in RREPs. These labels are then used to identify neighbors as viable loop-free successors to the destination, which allows intermediate nodes to generate replies. We call this approach Source-sequenced Labeled Routing (SLR).

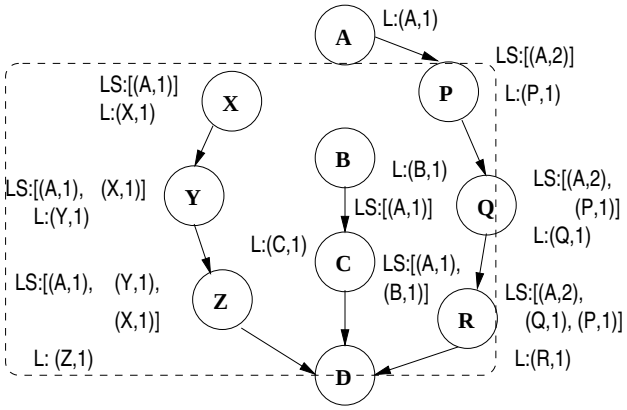
For the purposes of discussing SLR, we use the non-caching version of routing based on SSLs presented in Section 5.1.8. We later show how it can simplified to the caching version.

5.2.1 Viable Successor Sets



(a) AODV

(b) SLR using SSLs and RSLs



(c) SLR after reset

Figure 5.4: Viable Successor Set Dynamics

We consider the generalized class of on-demand hop-by-hop ad hoc routing protocols that use a RREQ flood identified by an unique SSL to build a tree rooted at the source, and label one or more of the reverse paths traversed by RREPs along the tree. We define a *viable*

successor set for a destination D at a node A at time t, denoted by $VSS_D^A(t)$, as the set of nodes that node A can use as a successor to destination D without causing any loops. We illustrate how VSS varies as a function of time when using a labeling scheme such as AODV.

Figure 5.4(a) shows a possible assignment of destination sequence numbers (as labels) when AODV is used as the routing protocol for a ten-node network, after node A attempts a route discovery for destination D and establishes a route at time t. Node A uses B as its next hop for destination D because it offers the shortest cost path. If at a later time t_1, link AB fails, then node A increases its destination-sequence number to two. Any new RREQs from node A can only be answered by a node that stores a destination-sequence number greater than or equal to two. Because of the labeling adopted by the nodes, VSS_D^A becomes empty ($\emptyset$) at time t_1 as none of the nodes in the shaded rectangle can satisfy the condition for generating a RREP. Consequently, RREQs from node A have to be answered by the destination.

5.2.2 Labeling with Source Sequence Numbers

To allow nodes with active routes to destination D to answer RREQs in SLR, each node stores a *label set* for the destination, which is an un-ordered collection of source-sequenced labels and relay-sequenced labels, together with its self source-sequenced label for the destination. The self source-sequenced label and label set for destination D maintained at node A are denoted by L_D^A and LS_D^A, respectively. The label set serves the purpose of other nodes identifying node A as a viable successor towards destination, while the the stored self sequenced-label is used by A to identify other nodes as safe viable successors. When routing state is lost, it is considered that $LS_D^A = \emptyset$ and $L_D^A = (A, \infty)$. Nodes can also choose to drop any of the elements

from their label sets for a destination at any time without affecting correct operation. We use the term sequenced-label (SL) to refer to a SSL or RSL.

We define the following operator ($\succeq$) to determine if a sequenced-label $SL_1 = (src_1, id_1)$ is fresher than another $SL_2 = (src_2, id_2)$ as follows:

$$SL_1 \succeq SL_2, \text{if } src_1 = src_2 \wedge id_1 \geq id_2$$

Labeling Rule: Let node n_i receive a RREP satisfying SSC for destination n_k carrying a list of SLs [(n_1, id_1), (n_2, id_2), ..., (n_{i-1}, id_{i-1}), (n_i, id_i), ..., (n_{k-1}, id_{k-1})], where (n_1, id_1) is the SSL and the other sequenced-labels are RSLs appended by relaying nodes. Node n_i performs the following steps:

- Node n_i must set $LS_{n_k}^{n_i}$=$LS_{n_k}^{n_i}$ - SL, if $\exists SL'$, such that $SL' \in$ RREP $\wedge$ $SL' \succeq SL$.
- Node n_i can assign itself a label set $LS_{n_k}^{n_i} \subseteq LS_{n_k}^{n_i} \cup \{(n_1, id_1), (n_2, id_2), ..., (n_{i-1}, id_{i-1})\}$.
- Node n_i can set $L_{n_k}^{n_i} = (n_i, id_i)$. However, if node n_i relays the RREP, it must set $L_{n_k}^{n_i} = (n_i, id_i)$ or to (n_i, ∞).

We denote the individual elements of the label L with src^L and id^L. RREPs carry a label set (LS) which is set to LS_D^A, where node A is transmitting the RREP for a destination D. The label set stored at node A for destination D reported by a neighbor B is denoted by LS_{DB}^A, and $ID(LS_I)_{DB}^A$ is used to represent the id_I of the SL (I, id_I) reported in the label set.

Source-Sequenced Labeling Condition (SSLC): Node A can switch successors to its neighbor B for destination D at time t, if it is true that $\exists$ L, such that $L \in LS_{DB}^A(t)$ and $L \succeq L_D^A(t)$ (i.e., $ID(LS_A)_{DB}^A(t) \geq id^{L_D^A}(t)$).

We establish the following lemmas (6, 7, and 8) for the labels stored at a node A when SSC is used for RREP processing, and *Rules 1 to 5* and the *labeling rule* are adhered to. We assume in Lemma 7 and 8 that there exists a node I in the network that is active or engaged in the route computation that node A is also engaged for.

Lemma 6. $id^{L_D^A}(t_1) \leq id^{L_D^A}(t_2)$, *where* $t_1 < t_2$.

Proof. Node A must change its label L_D^A for destination D only if it processes and relays a RREP. To accept a RREP at time t as per SSC, it must be true that the RREP carries a SL (A, id_A) such that $SID_D^A(t) \leq id_A$. Node A will re-label $L_D^A = (A, id_A)$, and can only accept RREPs carrying a SL (A, id), where $id > SID_D^A(t)$, for SSC to be satisfied. This is true even after reboots or state loss from Lemma 2. Hence, the value of $id^{L_D^A}$ is strictly non-decreasing with time, and the lemma is true. □

Lemma 7. $ID(LS_i)_D^A(t_1) \leq ID(LS_i)_D^A(t_2)$, *where* $t_1 < t_2$.

Proof. For $(I, id_I) \in LS_D^A$, it must be true that node A engaged in a route computation for which node I is active or engaged as well. The proof is by contradiction. Let (I, id_I^1) and (I, id_I^2) be two SLs such that $id_I^1 < id_I^2$, and $(I, id_I^2) \in LS_D^A$. We will now prove that node A cannot accept a RREP carrying a SL (I, id_I^1) and modify LS_D^A. Node A must have two SLs (A, id_A^1) and (A, id_A^2) engaged in a route computation along with node I's SLs to receive the RREPs; although the relation between id_A^2 and id_A^1 is not known. After accepting the RREP carrying a SL (I, id_I^2), node A must have set $SID_D^A > max(id_A^1, id_A^2)$. This is true even after reboots or state loss because of Lemma 2. When node A receives the RREP carrying a SL (I, id_I^1), it cannot satisfy SSC. Hence, node A will not modify its label set LS_D^A. Therefore, the

lemma is true. □

Lemma 8. *A correspondence (denoted by $\rightsquigarrow$) exists between the value of $ID(LS_I)_D^A$ reported at time t and the value of $id^{L_D^A}$ at time $t^- < t$, where t^- denotes the time when $ID(LS_I)_D^A$ was added to or modified in the label set LS_D^A.*

Proof. For $ID(LS_I)_D^A$ to be reported at time t, it must have been added to the label set at a time earlier than t when node A must have accepted a RREP satisfying SSC. Lemma 7 shows that the value of $ID(LS_I)_D^A$ does not decrease with time, and hence t^- must be the last time instant when $ID(LS_I)_D^A$ was modified or added to LS_D^A. As per labeling rules, there must exist a defined value for $id^{L_D^A}$ at time t^-, although it can be modified at a later time. Hence, the value of $ID(LS_I)_D^A$ reported at time t corresponds to $id^{L_D^A}$ at time t^-.

□

Theorem 31. *If nodes follow labeling rules and use SSLC to change successors, then no routing-table loops can be formed.*

Proof. Using the same argument as in Theorem 25, we derive the following inequalities along path $P_{ai} \subseteq P_{aD}$ when nodes follow labeling rules and use SSLC to switch successors. We use Lemmas 8 and 6.

$$id^{L^i_D}(t) \leq ID(LS_i)^i_{Da}(t) = ID(LS_i)^a_D(t_{s[2,old]}) \rightsquigarrow id^{L^a_D}(t^-_{s[2,old]}) \leq$$

$$id^{L^a_D}(t_{s[2,old]}) \leq id^{L^a_D}(t_{s[2,new]}) \leq ID(LS_a)^a_{Ds[2,new]}(t)$$

$$= ID(LS_a)^{s[2,new]}_D(t_{s[3,old]}) \rightsquigarrow ... \rightsquigarrow id^{L^{s[k,new]}_D}(t^-_{s[k+1,old]}) \leq$$

$$id^{L^{s[k,new]}_D}(t_{s[k+1,old]}) \leq id^{L^{s[k,new])}_D}(t_{s[k+1,new]}) \leq$$

$$ID(LS_{s[k,new]})^{s[k+1,new]}_{Ds[k,new]}(t) = ID(LS_{s[k,new]})^{s[k+1,new]}_D(t_{s[k+1,old]}) \rightsquigarrow$$

$$... \rightsquigarrow id^{L^{p[i]}_D}(t^-_{s[k+m,old]}) \leq id^{L^{p[i]}_D}(t_{s[k+m,old]}) \leq id^{L^{p[i]}_D}(t_i) \leq$$

$$ID(LS_{p[i]})^{p[i]}_{Di}(t) = ID(LS_{p[i]})^i_D(t_b) \rightsquigarrow id^{L_i}_D(t^-_b) id^{L_i}_D(t_b) \leq id^{L_i}_D(t)$$

This leads to the erroneous conclusion that $id^{L^i_D}(t) < id^{L^i_D}(t)$. Hence, no loops can be formed. □

5.2.3 Simplified Labeling

The disadvantage with SLR is that it requires maintaining huge label sets to identify viable successors. We simplify SLR's labeling scheme by making use of only the SSL of a RREP, and call it the Labeled Successor Protocol (LSR).

LSR can be derived from SLR using a subset of the original labeling rules as follows: When node A accepts a RREP satisfying SSC identified by SSL (S, ID_S) for destination D, it performs the following steps:

- Node A must set LS^A_D=$\{(S, ID_S)\}$.

- Node A can set L_D^A to (S, ID_S), if $S = A$. If the RREP is relayed, then node A must set $L_D^A = (A, \infty)$.

The two stored labels used can be replaced with one in the case of LSR because it can be seen clearly that only one of the two labels are useful for loop-free checks.

5.2.4 Example

Figure 5.4(d) shows an example of VSS dynamics using SLs. Source A starts a RREQ flood with SSL (A, 1), which gets relayed along paths XYZ and BC after the nodes forward it with their respective RSLs. The RREPs generated by the destination (for the purposes of this example, we assume the destination replies to all received requests) are processed and the nodes set their label sets LS as shown in the figure. There is no explicit ordering of nodes here, and despite the higher hop count of X, node A can still switch to X as a viable successor applying SSLC. Here, the $VSS_D^A(t) = \{X,Y,Z,B,C\}$. Note that, in-addition to node A, other nodes can also identify their respective viable successors for the destination. Assume that at time t_1, node A labels path PQR as viable successors as shown in Figure 5.4(e). Because node A does not relay the RREP, it can still retain its old L_D^A = (A,1). Despite switching to a new path, node A can still use all the old successors at a later time and the $VSS_D^A(t_1)$ is the complete set $\{B,C,X,Y,Z,P,Q,R\}$. Node A is able to determine all the nodes that were previously labeled as viable successors. However, as per the labeling rules, if node A had relayed the reply, it has to relabel L_D^A to $(A, 2)$, which will force A to lose viable successors from its old RREQ $(A, 1)$. The other reason the VSS can lose successors at a later time is if the relay nodes along the path drop SLs from their label set.

Figure 5.5(a) shows another example of the labeling for a network where source A has a route to destination D. At a later time, due to network mobility, node C's link to D fails. Node C re-establishes a route to D through a path $CEAFD$, where E and F are new nodes that are in this vicinity due to mobility. Figure 5.5(b) shows the labeling at this time. Subsequently, if node B's link to C fails, then node B establishes a new route through $BAFD$. Figure 5.5(c) shows the labeling at this time. Note that the re-labeling occurs in these cases because the RREP is generated by the destination. Note that in both these cases, the destination is the only node answering because node C or B cannot identify any viable successors. The label sets stored allow nodes to be identified as loop-free successors for a destination when SSLC is used. Figure 5.6 shows the labeling for the same set of events when LSR (using the simplified labeling scheme of SLR) is used as the routing protocol.

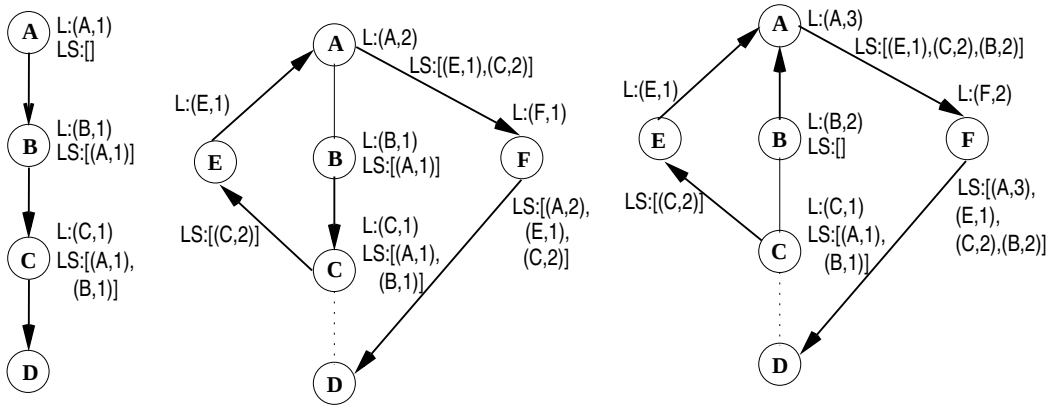


Figure 5.5: SLR labeling

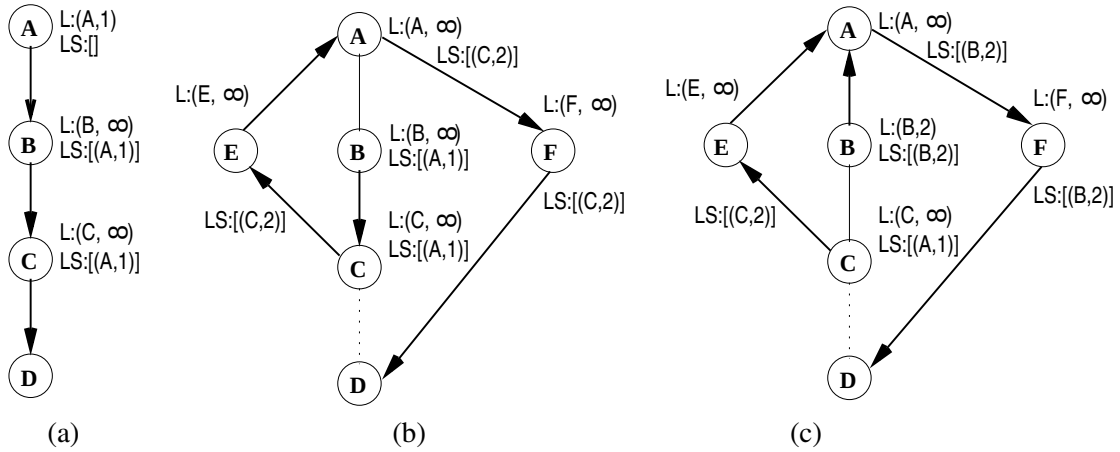


Figure 5.6: LSR labeling

5.2.5 Route Search

We present three conditions for SLR that are used by nodes to search for loop-free viable successors across a single-hop or multiple-hops to find a route to the destination.

We redefine the is-a-subset-of or equal-sets operation ($\subseteq$) between two label sets LS_1 and LS_2 as follows:

$$LS_1 \subseteq LS_2, \text{ if } \forall\, SL,\ SL \in LS_1, \text{it is true that}$$

$$\exists SL', \text{ such that } SL' \in LS_2\ \wedge SL' \succeq SL$$

Each RREQ carries a *common label set* CLS that represents the collection of self SLs of each node that transmitted the RREQ. The purpose of the CLS is to find a loop-free path from a successor whose RREP is usable at every node along the reverse path to the originating node. We use superscripts req and rep to represent the values carried in RREQs and RREPs, respectively. The conditions for initiating RREQs, relaying RREQs, and generating RREPs are as follows:

*R*LSC: (Reset Labeled Successor Condition). If node A must change s_D^A (after a route failure or if $s_D^A = \phi$), then it must send a RREQ carrying $CLS_D^{req} = L_D^A$.

*G*LSC: (Generate Labeled Successor Condition). Node I can issue a RREP responding to a RREQ req for destination D if I has an active route to D, and $CLS_D^{req} \subseteq LS_D^I$.

*C*LSC: (Common Labeled Successor Condition). When node A relays a RREQ for destination D, it sets $CLS_D^{req} = CLS_D^{req} \cup L_D^A$ in the relayed RREQ iff $CLS_D^{req} \subseteq LS_D^A$. Otherwise, CLS_D^{req} is set to $\emptyset$.

We illustrate an example of how the RREQ search progresses across multiple hops to find an intermediate node that can reply. Assume all nodes in Figure 5.4(e) except node R have expired their route for destination D. Node A sends a RREQ with $CLS_D^{req} = (A, 1)$. For simplicity we consider path PQR. Node P relays the RREQ with $CLS_D^{req} = [(A, 1), (P, 1)]$ as per CLSC because $(A, 1) \subseteq LS_D^P$. Similarly, node Q relays the RREQ with $CLS_D^{req} = [(A, 1), (P, 1), (Q, 1)]$. Now, GLSC allows node R to initiate a RREP that will satisfy SSLC at every one of the nodes Q, P, and A when the RREP traverses the reverse path.

When the simplified labeling scheme of SLR is used in LSR, nodes can only identify neighbors as successors. Route searches progressing more than a single-hop can only be answered by the destination.

5.2.6 Termination Properties

The proof that all nodes will invalidate their routing entries for a destination that is partitioned follows directly from Theorem 27 which shows that the property holds when the

destination is the only node that can generate replies. In SLR, according to Theorem 31, the DASG is instantaneously loop-free and no nodes ever choose any nodes upstream in the DASG. This means that the RERRs that propagate along the DASG will force all nodes to invalidate their route entries for the partitioned destination.

Theorem 32. *In an error-free stable connected network, a source will establish a route to a destination in finite time.*

Proof. Let source A issue a RREQ for destination n_1 that traverses a path $P = \{n_k, n_{k-1}, ..., n_i\}$ (where n_i can be n_1), before reaching the destination n_1 or a node that satisfies SLSC. If $CLS_{n_1}^{req} = \emptyset$, then the RREQ can only be answered by the destination, and the proof follows that of Theorem 27. Otherwise, when the RREP is transmitted along the reverse path to node n_{i-1} by node n_i, it is true that $LS_D^{rep} = LS_D^{n_i} \supseteq CLS_D^{req}$ because of GLSC and CLSC. Hence, node n_{i-1} must be able to accept the RREP, which will satisfy SSLC. The same argument holds at every node that relays the RREP along the reverse path to source A. If a node along path P modifies its label set by processing another route computation, then the RREP will not be accepted and will not be relayed to the source. However, sources retry RREQs and there are only a finite number of nodes in the network. From the same argument as in Theorem 28, each source must be able to establish a route to the destination. □

5.3 On-Demand Routing Using SSLs and Distance Information to Allow Replies from Nodes with Valid Routes

As the last component of our loop-free routing framework based on SSLs and RSLs, we present an approach with which nodes with valid routes to a destination are allowed to answer RREQs using SSLs and distance information rather than label sets, which may become large in some scenarios. In this alternative approach to SLR, which we call Labeled Source-sequenced Routing with Distances (LSR-D), distances are paired with SSLs to create a single label that we call *source-sequenced distance label* (SSDL). SSDL's create a relative ordering of the distances along the path in which nodes are engaged for a particular RREQ SSL, and the source of the SSL can identify all nodes in the path as viable successors regardless of the distances. We show that SSDLs can be safely used to maintain loop-freedom using an technique similar to the one presented for the prior approaches.

5.3.1 Sufficient Conditions for Loop-freedom

Each source-sequenced distance label (SSDL) is a tuple [(SSL), distance]. We now present the associated terminology, and operations on these labels. The freshness operator ($\succ$) and its inverse($\prec$), between two labels, $SSDL_1 = [(src_1, id_1), d_1]$, and $SSDL_2 = [(src_2, id_2), d_2]$ is defined as follows:

$$SSDL_1 \succ SSDL_2$$

$$\text{if } (src_1 = src_2 \ \wedge id_1 > id_2) \ \vee (src_1 = src_2 \ \wedge id_1 = id_2 \ \wedge d_1 < d_2) \quad (5.1)$$

$SSDL_1 \prec SSDL_2$

$$\text{if } (src_1 = src_2 \ \wedge id_1 < id_2) \ \vee (src_1 = src_2 \ \wedge id_1 = id_2 \ \wedge d_1 > d_2) \quad (5.2)$$

We denote the label stored for a known destination D at node A by $SSDL_D^A$. An invalid SSDL at a node A is considered to be [(A, ∞), 0]. The SSDL reported to node A by neighbor B for destination D is denoted by $SSDL_{DB}^A$. Each RREP carries the SSDL and distance metric (d) at the relaying node for the destination denoted by d_D^{rep}. We denote the cost of the link from node A to B with lc_B^A.

Distance Labeling Rule (DLR): When node A accepts a RREP from neighbor B for destination D that satisfies SSC and that is identified by SSL (S, ID_S), node A must set $SSDL_D^A$=$[(S, ID_S), d']$, where if S=A then $d' = \infty$ else $d' = d_D^{rep} + lc_B^A$. Node A can choose not to modify a valid $SSDL_D^A$ if it does not relay the RREP.

DLR allows nodes to assign or modify (reset) the stored SSDL for a destination. This may be done after the loss of state, or if the SSDL stored can no longer be used to determine any viable successors. Note that DLR requires SSC to be satisfied; therefore, nodes must still use the RSLs to determine which RREPs to accept and such RREPs must be generated by the destination. However, if nodes have valid SSDLs, they can choose a neighbor as a safe loop-free successor by comparing the freshness of the SSDLs as given by this sufficient condition for loop-freedom.

Source-Sequenced Distance Labeling Condition (SSDLC): Node A can switch successors to its neighbor B for destination D at time t, if it is true that $SSDL_{DB}^A(t) \succ SSDL_D^A(t)$.

Figure 5.7 shows the VSS for the same scenario discussed in Section 5.2.4 when SSDLs are used for labeling. We assume link costs to be unity. As the figure illustrates, a relay node B with SSDL [(A,1),2] can identify C with SSDL [(A,1),1] as a viable successor without using extensive label sets. Node A can still identify all nodes as viable successors, because its SSDL is set to the highest distance (∞). SSDLs also allow nodes to identify viable successors along different paths; for example, B can identify Y and Z.

We establish the following lemma for the SSDLC stored at a node A when SSDLC is used for RREP processing, and *Rules 1 to 5* and DLR are adhered to.

Lemma 9. $SSDL_D^A(t_1) \preceq SSDL_D^A(t_2)$, *where* $t_1 < t_2$.

Proof. This proof directly follows from Lemma 7 because node A derives (i.e., relabels) its SSDL, according to DLR, by associating the SSL of the RREP it accepts with the distance to the destination D. □

Theorem 33. *If nodes follow DLR and use SSDLC to change successors, then no routing table loops can form.*

Proof. Using the same argument as in Theorem 25, we derive the following inequalities along path $P_{ai} \subseteq P_{aD}$ when nodes follow DLR and use SSDLC to switch successors. We use Lemma

9.

$$SSDL_D^i(t) \prec SSDL_{Da}^i(t) = SSDL_D^a(t_{s[2,old]}) \preceq SSDL_D^a(t_{s[2,new]})$$

$$\prec SSDL_{Ds[2,new]}^a(t) = SSDL_D^{s[2,new]}(t_{s[3,old]})...SSDL_D^{s[k,new]}(t_{s[k+1,old]}) \preceq$$

$$SSDL_D^{s[k,new]}(t_{s[k+1,new]}) \prec SSDL_{Ds[k,new]}^{s[k+1,new]}(t) = SSDL_D^{s[k+1,new]}(t_{s[k+1,old]})$$

$$...SSDL_D^{p[i]}(t_{s[k+m,old]}) \preceq SSDL_D^{p[i]}(t_i) \prec SSDL_{Di}^{p[i]}(t) = SSDL_D^i(t_b) \preceq SSDL_D^i(t)$$

This leads to the erroneous conclusion that $SSDL_D^i(t) \prec SSDL_D^i(t)$. Hence, no loops can be formed. □

5.3.2 Route Search

Each RREQ carries the freshest of the SSDL's stored at the nodes along the path traversed by the RREQ, which is denoted by FSSDL. We define an in-order operator ($\sqcap$) between two SSDLs, $SSDL_1$ and $SSDL_2$, as follows:

$$\sqcap \quad (SSDL_1, SSDL_2) \qquad = \qquad \begin{cases} SSDL_2, \ if \ (SSDL_2 \succ SSDL_1) \\ \\ \phi, \ otherwise \end{cases}$$

We briefly describe conditions similar to that of SLR for nodes to search for routes across one or more hops and when intermediate nodes with valid routes can reply.

- RLSC-D: A node A issues a RREQ with $FSSDL_D^{req} = SSDL_D^A$ for destination D.
- CLSC-D: Node A relays RREQs with $FSSDL_D^{req} = \sqcap(FSSDL_D^{req}, SSDL_D^A)$.

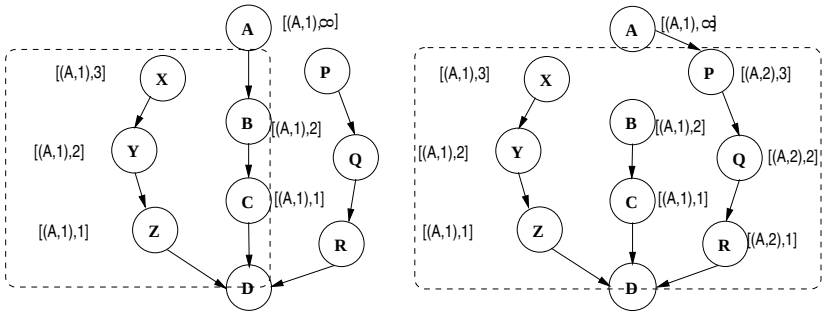
- GLSC-D: An intermediate node I having a valid active route can reply to a request if $SSDL_D^I \succ FSSDL_D^{req}$.

A *local-repair* operation can be performed by intermediate nodes to repair routes locally using a neighbor query that is a RREQ with *ttl* set to one. The in-order operator ($\sqcap$) is used when relaying RREQs, because a previously expired path to the destination can be activated without modifying the stored SSDLs. For example, in Figure 5.7, node A can search a route to D through a path PQR because the the SSDLs are in-order and every RREQ will be relayed with the stored SSDL. However, if the RREQ traverses a path PBC, then $FSSDL$ will be set to ϕ, and the RREQ can only be answered by the destination, which will force the nodes along the path to reset their SSDLs. A local-repair can be performed by an intermediate node B without sending a RERR to source A when its current link to C fails. A one-hop RREQ query sent will be answered by node Y or Q.

5.3.3 Termination Properties

The proof that all nodes will invalidate their routing entries for a destination that is partitioned follows directly from Theorem 27 which shows that the property holds when the destination is the only node that can generate replies. In LSR-D, according to Theorem 33, the DASG is instantaneously loop-free and no nodes ever choose any nodes upstream in the DASG. This means that the RERRs that propagate along the DASG will force all nodes to invalidate their route entries for the partitioned destination.

Theorem 34. *In an error-free stable connected network, a source will establish a route to a*



(a) Initial labeling (b) After reset

Figure 5.7: Labeling with SSDLs

destination in finite time.

Proof. Let source A issue a RREQ for destination n_1 that traverses a path $P = \{n_k, n_{k-1}, ..., n_i\}$ (where n_i can be n_1), before reaching the destination n_1 or a node that satisfies GLSC-D. If $SSDL_{n_1}^{req} = \phi$, then the RREQ can only be answered by the destination, and the proof follows that of Theorem 27. Otherwise, when the RREP is transmitted along the reverse path to node n_{i-1} by node n_i, it is true that $SSDL_D^{rep} = SSDL_D^{n_i} \succ FSSDL_D^{req}$ because of GLSC-D and CLSC-D. Hence, node n_{i-1} must be able to accept the RREP, which will satisfy SSDLC. The same argument holds at every node that relays the RREP along the reverse path to source A. If a node along path P modifies its label set by processing another route computation, then the RREP will not be accepted and will not be relayed to the source. However, sources retry RREQs and there are only a finite number of nodes in the network. From the same argument as in Theorem 28, each source must be able to establish a route to the destination. □

Table 5.1: Performance average over all pause times for 50 nodes network for 10-flows and 30-flows

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
DSLR	10	0.996±0.001	0.016±0.002	0.271±0.067	2.612±0.182
LSR	10	0.995±0.001	0.017±0.002	0.313±0.082	2.628±0.186
LSR-D	10	0.995±0.001	0.025±0.005	0.374±0.093	2.535±0.167
LSR-D-LR	10	0.995±0.001	0.025±0.004	0.372±0.098	2.539±0.168
AODV	10	0.994±0.002	0.016±0.003	0.270±0.066	2.576±0.179
AODV-LR	10	0.994±0.002	0.017±0.004	0.266±0.067	2.580±0.180
DSR	10	0.940±0.027	0.041±0.047	0.220±0.095	2.677±0.185
OLSR	10	0.887±0.040	0.012±0.001	1.937±0.220	2.456±0.175
DSLR	30	0.830±0.035	0.446±0.100	3.566±0.821	2.617±0.172
LSR	30	0.859±0.038	0.480±0.170	2.229±0.657	2.678±0.223
LSR-D	30	0.858±0.037	0.475±0.159	2.226±0.628	2.675±0.224
LSR-D-LR	30	0.867±0.036	0.442±0.159	1.777±0.507	2.703±0.229
AODV	30	0.765±0.055	1.010±0.356	4.423±1.289	2.951±0.324
AODV-LR	30	0.770±0.056	0.965±0.333	4.269±1.264	2.929±0.309
DSR	30	0.683±0.059	4.760±1.073	0.410±0.140	3.625±0.308
OLSR	30	0.798±0.034	0.883±0.311	0.713±0.069	2.478±0.161

Table 5.2: Performance average over all pause times for 100 nodes network for 10-flows and 30-flows

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
DSLR	10	0.991±0.004	0.034±0.006	0.907±0.237	3.851±0.307
LSR	10	0.990±0.004	0.042±0.007	1.192±0.342	3.814±0.314
LSR-D	10	0.989±0.005	0.069±0.013	1.462±0.412	3.690±0.302
LSR-D-LR	10	0.988±0.005	0.069±0.013	1.353±0.405	3.747±0.294
AODV	10	0.988±0.004	0.036±0.009	0.897±0.236	3.744±0.293
AODV-LR	10	0.988±0.004	0.035±0.008	0.872±0.221	3.767±0.293
DSR	10	0.876±0.050	0.099±0.057	0.859±0.353	4.257±0.317
OLSR	10	0.821±0.063	0.022±0.002	11.795±1.575	3.583±0.256
DSLR	30	0.690±0.033	0.728±0.107	11.845±1.733	3.864±0.194
LSR	30	0.737±0.039	0.751±0.129	8.213±1.473	3.941±0.237
LSR-D	30	0.738±0.039	0.754±0.133	8.248±1.503	3.961±0.244
LSR-D-LR	30	0.757±0.033	0.669±0.118	6.229±1.154	4.020±0.259
AODV	30	0.608±0.051	1.455±0.385	18.298±13.069	4.751±0.434
AODV-LR	30	0.592±0.044	1.617±0.538	21.339±15.523	4.868±0.463
DSR	30	0.618±0.049	5.125±0.782	1.243±0.405	6.141±0.499
OLSR	30	0.612±0.041	3.371±0.532	5.423±0.669	4.014±0.277

Table 5.3: Performance average over all pause times for 50-nodes and 100-nodes network with 30-flows (fixed, long-lived)

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
DSLR	50	0.803±0.053	0.432±0.148	4.027±1.113	2.716±0.199
LSR	50	0.823±0.053	0.444±0.185	3.194±1.006	2.754±0.224
LSR-D	50	0.842±0.059	0.458±0.251	2.250±0.864	2.797±0.288
LSR-D-LR	50	0.847±0.058	0.450±0.252	1.845±0.751	2.840±0.300
AODV	50	0.738±0.083	0.866±0.473	5.044±1.874	3.084±0.424
DSR	50	0.581±0.081	3.422±0.813	0.430±0.156	3.809±0.406
OLSR	50	0.772±0.042	0.842±0.413	0.727±0.072	2.534±0.187
DSLR	100	0.686±0.063	0.589±0.143	11.627±2.837	3.913±0.286
LSR	100	0.695±0.063	0.641±0.169	10.934±2.470	4.016±0.372
LSR-D	100	0.744±0.064	0.567±0.189	7.304±2.048	4.042±0.388
LSR-D-LR	100	0.769±0.062	0.496±0.176	5.289±1.691	4.103±0.401
AODV	100	0.608±0.088	1.060±0.402	16.812±8.743	4.731±0.696
DSR	100	0.476±0.099	3.865±1.150	1.208±0.453	6.177±0.733
OLSR	100	0.585±0.066	2.761±1.062	5.541±0.811	4.074±0.452

5.4 Simulation Results

We present results over varying loads and mobility for an instantiation of DSLR that selects shortest-cost paths, LSR, which adopts the simplified labeling scheme of SLR, and LSR-D, which uses SSDLs. We also present results for LSR-D-LR, which is LSR-D with the local-repair scheme. The protocols used for comparison are two on-demand protocols, DSR and AODV, and OLSR, which is a pro-active link state protocol. Simulations are run in Qualnet 3.5.2. AODV, DSLR, LSR, LSR-D, and LSR-D-LR, use an expanding-ring search scheme when flooding RREQs. AODV and DSLR set the ttl of the RREQs to the last-known hop-count of the destination.

Simulations are performed for two scenarios, (i) a 50-node network with terrain dimensions of 1500m x 300m, and (ii) a 100-node network with terrain dimensions of 2200m x 600m. Traffic loads are CBR sources with a data packet size of 512 bytes. Load is varied by

using 10 flows (at 4 packets per second) and 30 flows (at 4 packets per second). The MAC layer used is 802.11 with a transmission range of 275m and throughput 2 Mbps. The simulation is run for 900 seconds. Node velocity is set between 1 m/s and 20 m/s. We have two sets of traffic flow patterns: (i) flows that have a mean length of 100 seconds distributed exponentially between randomly picked sources and destinations; and (ii) flows between fixed pairs of sources and destinations that last the entire simulation time. Each combination (number of nodes, traffic flows, scenario, routing protocol and pause time) is repeated for nine trials using different random seeds.

We present four metrics. *Delivery ratio* is the ratio of the packets delivered per client/server CBR flow. *Latency* is the end to end delay measured for the data packets reaching the server from the client. *Network load* is the total number of control packets (RREQ, RREP, RERR, Hello, TC etc) divided by the received data packets. *Data hops* is the number of hops traversed by each data packet (including initiating and forwarding) divided by the total received packets in the network. This metric takes into account packets dropped due to forwarding along incorrect paths. A larger value for the data-hops metric indicates that more data packets traverse more hops without reaching the destination necessarily.

5.4.1 Performance Summary

Tables 5.1 and 5.2 summarize the results of the different metrics by averaging over all pause times for the 50 and 100 node networks with random flows. Table 5.3 summarizes the same set of metrics for 50 and 100-node networks with fixed flows. The columns show the mean value and 95% confidence interval. All our performance discussions focus on the average

case because the confidence intervals overlap atleast slightly in most cases. The packet delivery ratio, the end-to-end delay, and the control overhead over various pause times is shown for a 100-node network with 30-flows in Figures 5.8, 5.9 (results for LSR-D are not shown). The vertical bars in the graphs indicate the 95% confidence intervals.

The performance results show that DSLR, LSR, and LSR-D outperform AODV, DSR, and OLSR. LSR has better packet delivery and lower control-overhead than DSLR across the different scenarios. This is because the labeling in LSR allows one-hop neighbors of the source to initiate RREPs, thus avoiding RREQ floods, whereas RREPs in DSLR can be initiated by the destination only. Note that if the elaborate labeling scheme of SLR is used, it is also possible for intermediate nodes that are across multiple hops from the source to initiate RREPs. The latency of DSLR is slightly better than of LSR because DSLR establishes more optimal (shortest-cost) paths because RREQ floods search for the destination after a route failure. In the case of LSR, the replies from intermediate nodes need not necessarily reflect the best path when nodes are mobile. The optimal forwarding of packets is also shown in the data-hops metric of DSLR, which is slightly lower than that of LSR. LSR-D's performance is equivalent to that of LSR across the different scenarios. However, the local-repair of LSR-D-LR shows a noticeable improvement in performance, particularly, in the 100-node, 30-flow scenarios where excessive RREQ flooding can cause congestion. The local-repair allows nodes to resolve failure without reporting a RERR to the source, which will then retry flooding RREQs. The key feature is that the RREQ sent with a *ttl* of one can elicit a reply from one of the neighbors. Although, AODV supports a local-repair operation, it floods the RREQ to locate the destination.

The performance of AODV, DSR, and OLSR suffer from the following problems:

AODV suffers from excessive RREQ flooding. Note, however, that DSLR must also have RREPs sent by the destination, making it almost similar to AODV's case. We noticed that AODV was generating an excessive number of RREPs, and yet the number of RREPs received at the sources was far smaller. This could be caused by RREPs being forwarded by nodes using a reverse route entry, which is only valid for a limited time. Hence, with congestion, it is possible that these RREPs get delayed and are dropped. DSR suffers from stale caches and source-routes need not necessarily reflect the topology of the mobile network. OLSR suffers from temporary loops. The number of messages exchanged in OSLR is constant although the control overhead calculated depends on the number of flows in the network.

5.4.2 Performance Improvements with Local route recovery

We compare the performance of the LSR-D one-hop route recovery mechanism against other proposed mechanisms. We simulate AODV with the local route repair operation as specified in the draft, in the same set of scenarios. The version also includes the expanding ring search as discussed in [33]. Although, DSR can recover routes locally using a *packet salvaging mechanism*, we could not simulate it because a a *salvage count* is required to prevent data packets from being recovered and sent in loops infinitely, and an ideal salvage count value is not specified in the draft.

The summarized results for AODV-LR (AODV with local repair) are listed in Tables 5.1, and 5.2. The performance results for AODV-LR show no significant improvement over AODV. Table 5.4 shows the number of attempts and successful local repairs for LSR-D-LR and AODV in a 100-node network with 30-flows for all pause times. The statistics show that

Table 5.4: Local Repair statistics for 100 nodes network with 30-flows

	AODV-LR		LSR-D-LR	
Pause Time	RREQ Attempts	%Repair	RREQ Attempts	%Repair
0	81.33	91.26	3684.11	37.15
50	90.44	90.42	3582.67	36.96
100	108.11	90.24	3830.89	36.77
200	151.33	92.51	3860.44	38.51
300	205.89	92.66	3672.11	41.13
500	190.67	92.42	3333.33	43.02
700	211.00	92.89	3565.89	42.79
900	200.56	93.68	3221.89	45.19

AODV only attempts about 100-200 local repairs and $\tilde{9}0\%$ of them succeed (by flooding to the destination). In the case of LSR-D-LR, about 4000 local repairs are attempted and half of them succeed. The reason for the low number of RREQ attempts by AODV is that the specification states that the local repair RREQ flood must not reach the source. Therefore, a check is made on the ttl of a local repair RREQ; if it will reach the source, then the repair attempt is aborted. LSR-D-LR suffers no such limitation as it performs only performs a one-hop neighbor query. These set of simulations were run on asymmetric flow distributions, which means that the AODV local repair operation might fail because it does not know the hopcount to the source. However, our results correlate with the recent study [33] on the scalability of AODV. The results are presented for a set of 20 randomly picked source/destination CBR flows(unidirectional), in networks with up-to 10,000 nodes under different optimizations. However, in networks of 50 to 100 nodes, it can be deduced ([33], Fig.6, pp.106) that the number of local route repair attempts is of the order of 100-200, correlating with our results.

To validate our intuition here, we ran simulations for AODV with local repair and LSR-D-LR for a network of 50 nodes, with 15 random bi-directional flows, exponentially dis-

tributed at 100 secs; the average performance results are tabulated in Table 5.5. LSR-D-LR, once again, outperforms AODV by a huge margin. Table 5.6 shows the local repair statistics for AODV-LR, and LSR-D-LR in the scenario with bi-directional flows. LSR-D-LR recovers about half of the route failures with a neighbor query, after sending about 2500 local repair RREQs. On the other hand, AODV's local repair scheme only initiates about less than ten RREQs per run. This is about ten times less than the previous results for the 100-node scenarios. The *skipped* column shows the average number of skipped attempts in each pause time which is about every attempt. The reason for so many skipped attempts is because of the ttl-check for the local repair RREQ [33]: A local repair RREQ *est-ttl* is calculated to fall in the range, AODV_MIN_ REPAIR_TTL $<$ *est-ttl* $<$ (hopcount-to-source)/2 + AODV_LOCAL_ADD_TTL. The prescribed value for AODV_LOCAL_ADD_TTL is two, and that for AODV_MIN_REPAIR_TTL is one. To initiate a RREQ, it is necessary that *est-ttl* be less than the *hopcount-to-source*. Solving the inequality, a node can initiate a RREQ only if (hopcount-to-source) $>$ 4. This basically limits the route repair operation to only intermediate nodes which are atleast at a distance of four hops from the source; if the total path length is less than four hops, no attempts are made. This also explains the disparity of very few attempts in the 50-nodes network compared to more local repair attempts in the 100-nodes network, because of longer path lengths.

Table 5.5: Performance average over all pause times for bi-directional flows

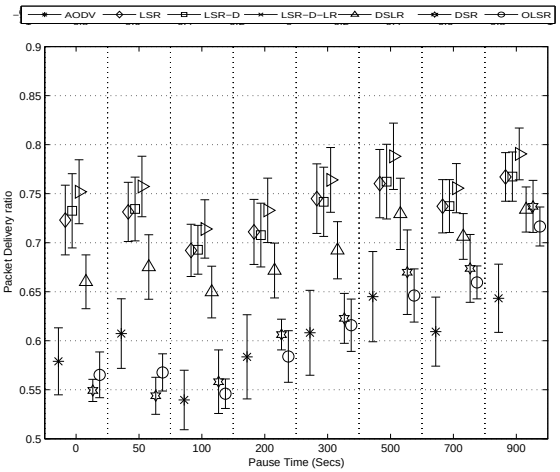
Metric	LSR-D-LR	AODV-LR
Delivery Ratio	0.828±0.032	0.699±0.074
Latency (sec)	0.618±0.178	1.449±0.467
Net Load	2.378±0.558	4.911±1.265
Data Hops	2.790±0.215	3.158±0.447

Table 5.6: Local Repair statistics for bi-directional flows

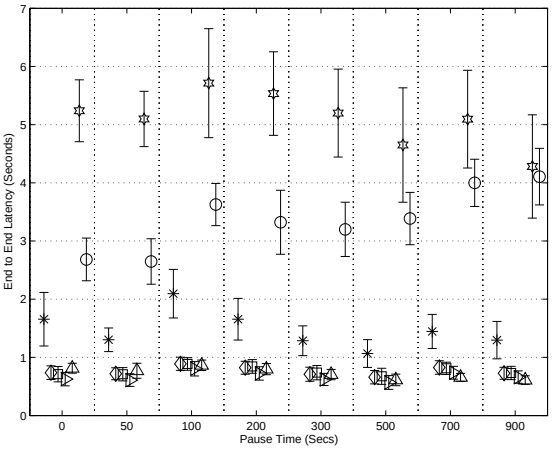
	AODV-LR			LSR-D-LR	
Pause Time	Skipped	RREQs	%Repair	RREQs	%Repair
0	2201.89	3.33	90.00	2003.12	38.21
50	2510.67	5.44	95.92	2088.33	38.02
100	2844.89	7.78	95.71	2341.00	37.28
200	2603.44	4.89	100.00	2205.78	37.74
300	2994.67	6.56	100.00	2523.78	37.88
500	2938.78	6.33	96.49	2436.11	40.59
700	3349.67	7.00	96.83	2908.78	36.29
900	2959.56	5.00	100.00	2512.89	40.77

5.5 Conclusion

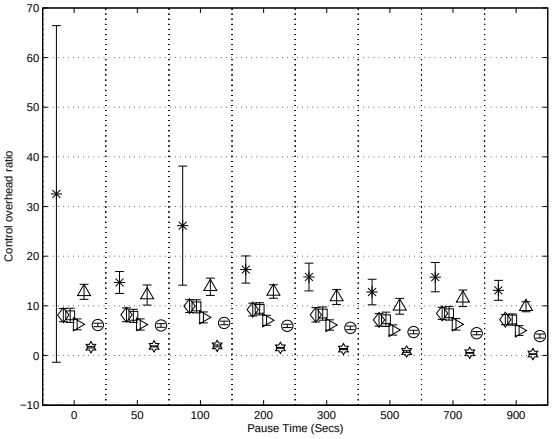
We have introduced the first routing framework for on-demand loop-free routing in MANETs based on the source sequence number that is used to identify RREQs originated uniquely. We presented three approaches within this framework (DSLR, SLR, and LSR-D) that are robust to node failures, loss of information, and unreliable message delivery. They allow hop-by-hop routing of data packets while maintaining instantaneous loop-freedom of the routing tables without requiring time-stamps, source-routes, or any other synchronization techniques spanning single (i.e., packet filtering) or multiple hops. Performance results in 50 and 100-node networks with random traffic flows show that protocol instantiations of the proposed frameworks consistently deliver more data packets than DSR, AODV, and OLSR, while reducing control overhead and data packet latency.



(a) Packet Delivery

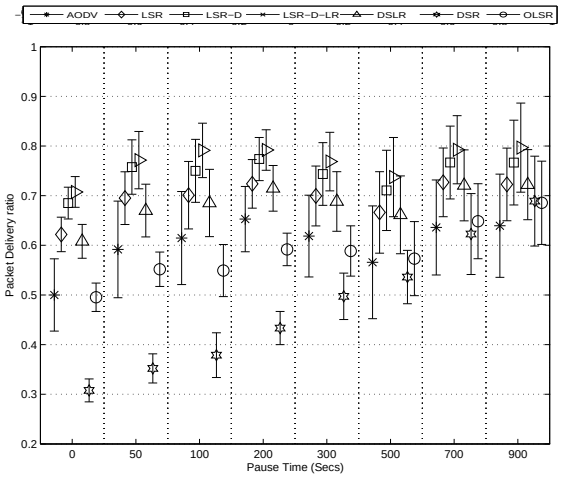


(b) Latency

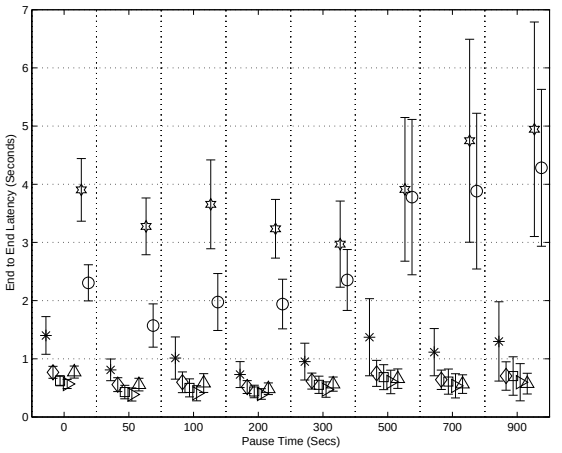


(c) Control Overhead

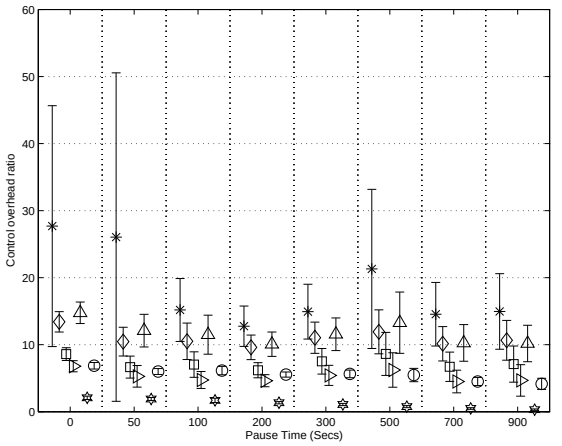
Figure 5.8: Random (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

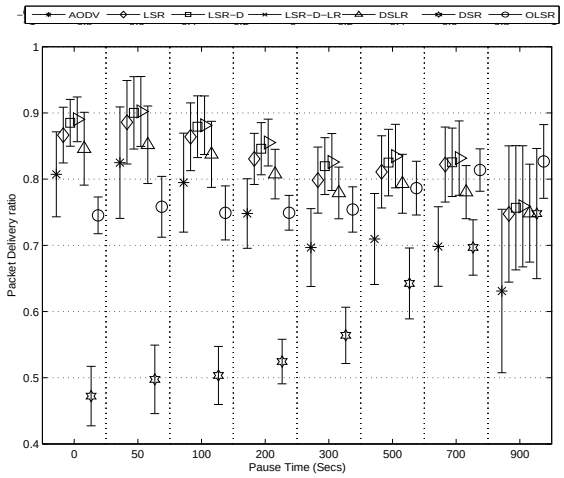


(b) Latency

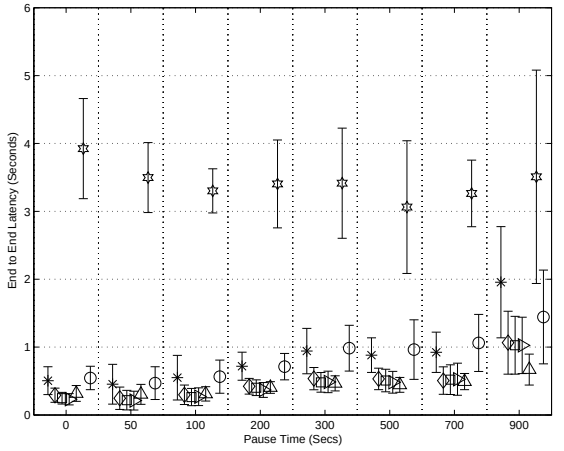


(c) Control Overhead

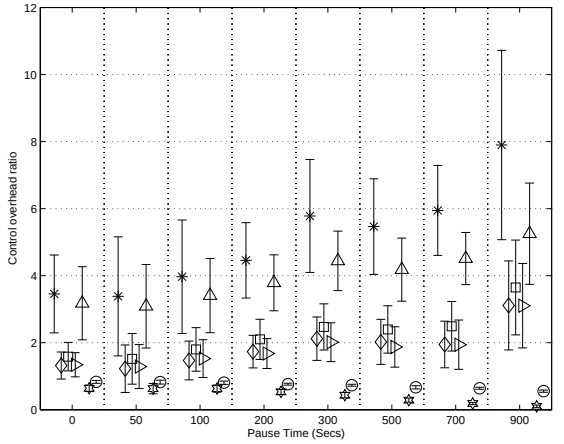
Figure 5.9: Fixed (100-nodes, 30-flows, 120 pps)



(a) Packet Delivery

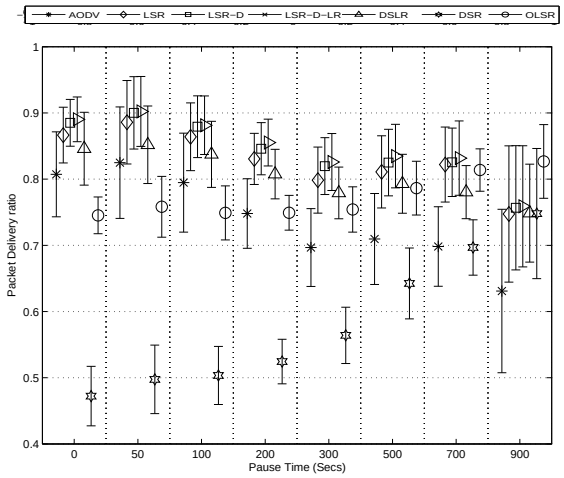


(b) Latency

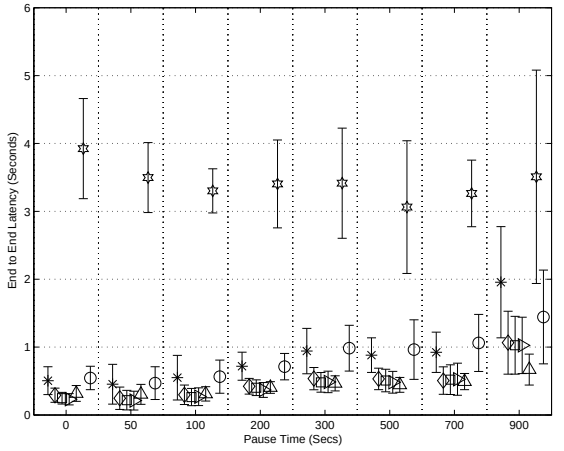


(c) Control Overhead

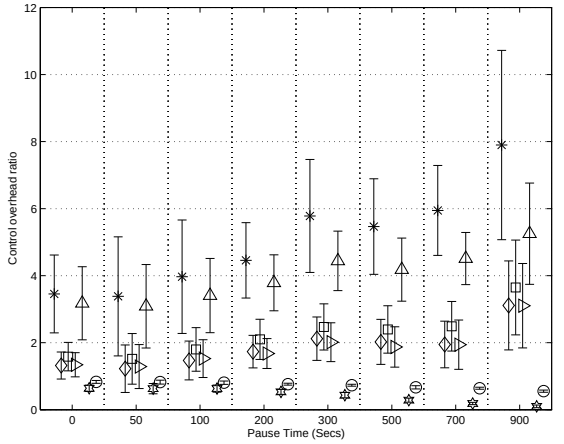
Figure 5.10: Random (50-nodes, 30-flows, 120 pps)



(a) Packet Delivery



(b) Latency



(c) Control Overhead

Figure 5.11: Fixed (50-nodes, 30-flows, 120 pps)

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we explore new robust and efficient techniques for loop-free on-demand routing in mobile ad hoc networks (MANETs). Current on-demand proposals attain loop-freedom by basing their routing decisions using either the freshness of updates determined using timestamps, or with topology information. Using only the same information collected as in prior proposals, we propose more effective mechanisms to attain loop-freedom while still improving performance. We prove the correctness of our protocols even when operating in MANET environments that are characterized by unreliable message delivery and failure of nodes. We show with extensive simulation experiments that our proposed protocols perform better than the current state-of-the-art MANET protocols prescribed in the IETF working group.

In the context of destination-sequence number based protocols, we show how the sequence-number handling in current protocols can cause temporary loops, defacto partitions,

and count-to-infinity. We present a new destination-sequence number framework that eliminates these problems and show how it can be applied to previous proposals to solve their problems. Differing from the traditional notion of a sequence number as a timestamp, we show how sequence numbers can be manipulated to achieve better performance in an on-demand routing protocol.

In topology-based routing protocols, the fundamental problem associated with maintaining loop-freedom is that of ensuring consistent information across the network when making routing decisions. We present two different approaches to solving this problem. In the first approach, we translate path information, which may become inaccurate later, into labels that are stored at nodes to maintain a strict "lexicographic" ordering that become "smaller" along any successor path to the destination. By allowing nodes to pick only successors that have a "smaller" label, loops are prevented. We illustrate how packet-filtering or our destination-sequence number framework can be used by nodes to reset stored labels. In the second approach, we introduce the notion of trusted topology information. Nodes maintain a list of untrusted links that should not be used for their routing decisions, inaddition to their known trusted links to a destination. This allows nodes to make routing decisions based on correct topology information, which inherently allows loop-free operation. We use the destination-sequence number mechanism as a reset to allow nodes to trust all links.

All on-demand routing protocols proposed to date use a uniquely identified flood search that creates loop-free paths to the destination. However, all of them use extra information and mechanisms to attain loop-free routing. We motivate our research by asking if loop-free routing can be achieved using the most minimal information that is required in any on-demand

routing protocol for a flood search. We present the first framework that allows loop-free routing using these same paths that are created during the flood search. To improve the performance of the protocol, we show how additional information can be collected within the same flood search framework and be used effectively to recover routes.

6.2 Future Work

In this thesis, we have focussed on the loop-freedom and safety of the routing protocols. The generic loop-free conditions we present can be used in the context of wired routing protocols, i.e., feasible labels used in FLR can be extended for use in the Border Gateway Routing (BGP) protocol to obtain instantaneous loop-freedom. The loop-free techniques can also be applied to multipath and multicast routing.

Energy efficiency is an important consideration for MANETs; therefore, it will be interesting to investigate the energy efficiency of the proposed protocols, given that all of them incur less control overhead. Quality of Service (QoS) routing takes importance in MANETs that require support for real-time traffic. An interesting extension to the currently proposed protocols would be to support QoS based routing.

Security is an important issue in the context of ad hoc networks where malicious intruders can join and disrupt routing in the network. Most secure routing protocol proposals require the destination to answer the route request with a route reply which can be verified along the successor path being set up. The source sequence-number routing framework can be used in this context to design a secure routing protocol. The SSLs and RSLs can be encrypted and

because they are only stored and relayed by nodes other than the ones originating them, the integrity of the routing messages can be verified.

Bibliography

[1] I.D. Chakeres and L. Klein-Berndt, "AODVjr, AODV Simplified," *ACM Mobile Computing and Commun. Review*, Vol. 6, No. 3, 2002, pp. 100-101.

[2] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," Request for Comments 3626, October 2003.

[3] M. Spohn and J. J. Garcia-Luna-Aceves, "Neighborhood Aware Source Routing," *Proc. ACM MobiHoc 2001*, pp. 11–21, Long Beach, CA, Oct. 4–5, 2001.

[4] H. Rangarajan, and J. J. Garcia-Luna-Aceves, "Achieving Loop-Free Incremental Routing in Ad Hoc Networks," Proc. ISCC 2004, Alexandria, Egypt, July 2004

[5] H. Rangarajan and J. J. Garcia-Luna-Aceves, "Loop-free On-demand Routing using Source Sequence Numbers", The 2nd IEEE MASS, Nov 7-11, 2005, Washington D.C., Washington, USA.

[6] E. M. Gafni and D. P. Bertsekas, "Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology," *IEEE Trans. Comm.*, COM-29(1):11–18, Jan. 1981.

[7] J. J. Garcia-Lunes-Aceves, "Loop-Free Routing Using Diffusing Computations," *Proc. IEEE/ACM Trans. Networking*, 1(1):130–41, Feb. 1993.

[8] J. J. Garcia-Luna-Aceves, M. Mosko and C. Perkins, "A New Approach to On-Demand Loop-Free

Routing in Ad Hoc Networks," *Proc. PODC 2003*, pp. 53-62, Boston, Massachusetts, July 13–16, 2003.

[9] M. Gerla, X. Hong, and G. Pei, "Fisheye State Routing Protocol (FSR) for Ad Hoc Networks," draft-ietf MANET-fsr-02. txt, IETF MANET Working Group-Internet Draft, Dec. 2001.

[10] J. J. Garcia-Luna-Aceves and M. Spohn, "Source-Tree Routing in Wireless Networks," *Proc. IEEE ICNP'99*, pp. 273–82, Toronto, Canada, October 31–November 3, 1999.

[11] S. Gwalani, E. M. Belding-Royer, and C. E. Perkins, "AODV-PA: AODV with Path Accumulation," *Proc. Next Generation Internet Symposium*, Anchorage, Alaska, May 2003.

[12] Y. C.Hu and D. B. Johnson, "Implicit Source Routing in On-Demand Ad Hoc Network Routing," In *ACM MOBIHOC 2001*, pp. 1-10, Long Beach, CA, Oct. 4–5, 2001.

[13] D. Johnson et al, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," IETF Internet draft, draft-ietf-manet-dsr-09.txt, April 2003.

[14] J. J. Garcia-Luna-Aceves and H. Rangarajan, " A New Framework for Loop-Free On-Demand Routing Using Destination Sequence Numbers," The 1st IEEE Conference on Mobile and Ad hoc Sensor Systems, October 25-27, 2004, Fort Lauderdale, Florida, USA.

[15] H. Rangarajan and J. J. Garcia-Luna-Aceves,"Making On-demand Routing Protocols Based on Destination Sequence Numbers Robust," Proc.ICC, May 16-20,2005, Seoul, Korea.

[16] S. Gwalani, E. M. Belding-Royer and C. E. Perkins, "AODV-PA: AODV with path accumulation," *ICC 2003 - IEEE International Conference on Communications*, vol. 26, no. 1, May 2003, pp. 527 - 531.

[17] R. Ogier et al., "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF),"Request for Comments 3684, February 2004.

[18] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," In *IEEE INFOCOM*, pp. 1405–13 vol.3, Apr. 1997.

[19] C. Perkins et al., "Ad hoc On-Demand Distance Vector (AODV) Routing," Request for Comments 3561, July 2003.

[20] S. Roy and J. J. Garcia-Luna-Aceves, " Using Minimal Source Trees for On-Demand Routing in Ad Hoc Networks," *Proc. IEEE INFOCOM 2001*, Anchorage, Alaska, April 22-26, 2001.

[21] C. Sengul, "Local Route Recovery in Mobile Ad Hoc Networks," M.S. Thesis, Computer Science, Univ. of Illinois at Urbana-Champaign, 2003.

[22] M. Spohn and J. J. Garcia-Luna-Aceves, "Neighborhood Aware Source Routing," *Proc. ACM MobiHoc 2001*, pp. 11–21, Long Beach, CA, Oct. 4–5, 2001.

[23] S. Vutukury and J.J. Garcia-Luna-Aceves, " A Simple Approximation to Minimum-Delay Routing, " *Proc. ACM SIGCOMM '99*, Cambridge, Massachusetts, September 1–3, 1999.

[24] K. Bhargavan, C.A. Gunther, and D. Obradovic, "Fault Origin Adjudication," *Proc. Workshop on Formal Methods in Software Practice*, Portland, OR, Aug. 2000.

[25] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Proc. ACM SIGCOMM 94*, pp. 234 – 244, 1994, London, United Kingdom.

[26] J. J. Garcia-Luna-Aceves and H. Rangarajan, "A New Framework for Loop-Free On-Demand Routing Using Destination Sequence Numbers," *Proc. IEEE International Conference on Mobile Ad-hoc and Sensor Systems* (MASS), October, 2004, Fort Lauderdale, Florida, USA.

[27] M. Mosko and J. J. Garcia-Luna-Aceves, "Loop-Free Routing Using a Dense Label Set in Wireless Networks," *Proc. ICDCS 2004*, March, 2004, Tokyo, Japan.

[28] R. V. Boppana and S. P. Konduru, “An Adaptive Distance Vector Routing Algorithm for Mobile, Ad Hoc Networks,” *Proc. IEEE Infocom*, Volume 3, pp. 1753–1762, Anchorage, Alaska, April 2001.

[29] M. K. Marina and S. R. Das. “On-Demand multipath distance vector routing for ad hoc networks,” *Proc. of the Int’l Conf. for Network Procotols (ICNP)*, pp. 14–23, Riverside, 2001.

[30] C. E. Perkins, E. M. Belding-Royer and I. D. Chakeres, “Ad hoc On-Demand Distance Vector (AODV) Routing,” IETF Internet Draft, draft-perkins-manet-aodvbis-01.txt, January 2004.

[31] E. M. Royer and C. E. Perkins, “Multicast Operation of the Ad hoc On-Demand Distance Vector Routing Protocol,” *Proc. MobiCom ’99*, pp. 207–218, Seattle, WA, August 1999.

[32] M. Chandra, “Extensions to OSPF to Support Mobile Ad Hoc Networking,” IETF Internet draft, draft-chandra-ospf-manet-ext-02, October 2004.

[33] S. J. Lee, E . M. Belding-Royer and C. E. Perkins, “Scalability study of the ad hoc on-demand distance vector routing protocol,” *Int. Journal on Network Management*, 13(2), 2003, 1099-1190, pp. 97–114.

[34] H. Rangarajan and J.J. Garcia-Luna-Aceves, “Using Labeled Paths for Loop-free On-Demand Routing in Ad Hoc Networks,” *Proc. ACM MobiHoc 2004*, Tokyo, Japan, May 24–26, 2004.

[35] ”IEEE standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” Nov 1997, P802.11.

[36] J. Raju and J. J. Garcia-Luna-Aceves, “A New Approach to On-Demand Loop-Free Multipath Routing,” *Proc. IEEE IC3N’99*, pp. 522–7, Boston, Massachusetts, Oct. 11–13, 1999.

[37] V. Park and S. Corson, ”A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Network,” *Proc. INFOCOM ’97*, pp. 1405–13, Washington DC, USA, 1997.

[38] E. B. Royer, I. Chakeres, D. B.Johnson, and C. E.Perkins, "Dynamic MANET On-demand Routing Protocol (DyMO)", IETF Internet draft, draft-ietf-manet-dymo-00, January 2005.

[39] J. Behrens and J.J. Garcia-Luna-Aceves, "Fast Dissemination of Link States Using Bounded Sequence Numbers with no Periodic Updates or Age Fields," *Proc. ICDCS'97*, Baltimore, Maryland, May 27–30, 1997.

[40] Y. C. Hu and D. B. Johnson, "Ensuring Cache Freshness in On-Demand Ad Hoc Network Routing Protocols," *Proc. of the POMC 2002 Workshop*, pp. 25-30, ACM, Toulouse, France, October 2002.

[41] C. Perkins et al, "Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks," *IEEE Personal Communications*, 8(1):16 – 28, Feb 2001.

[42] IETF MANET Working Charter, http://www.ietf.org/html.charters/manet-charter.html.

[43] J. J. Garcia-Luna-Aceves and S. Roy, "On-Demand Loop-Free Routing with Link Vectors," *12th IEEE International Conference on Network Protocols* (ICNP'04), Berlin, Germany, October 5-8, 2004.

[44] Scalable Network Technologies. Qualnet 3.5.2.